Introduction to JavaScript

JAVA !=JavaScript

REVATURE

Scripting Language

REVATURE

- Humble beginnings
  - Not meant for creating desktop application
  - Works only in web browser
  - Can not use JS for accessing local files, databases and USB files
- Built mainly for,
  - DOM manipulation
  - Client side language (runs on browser)
- However, because of popularity it has evolved into desktop (adobe photoshop) and server side products (node.js)

What is JavaScript

REVATURE

- An object oriented, dynamic language that has become the lingua franca of websites, but isn't limited to use on web pages
- Object oriented
  - Prototypical inheritance instead of class based
- Dynamic
  - Dynamically typed (Loosely typed) – variable types are interpreted at runtime
- Non-Compiled
  - Just in time compiled language (JIT, embedded in browser)

JavaScript Introduction

REVATURE

- Ease of use
- Interpreted from source code
- Relatively Loose Structure, actually heavily structured from ideas in C
- Function level scope rather than block level scope
- If javascript is disabled in browser, the javascript code won’t work

Editors

REVATURE

- Mac
  - Bare Bones
  - Brackets
  - Coda
- Mac / Windows
  - ATOM
  - Sublime
  - Aptana
- Windows
  - Web Storm (IDE)
  - Eclipse
  - Notepad++

Hello JavaScript

REVATURE

- Good practices doesn't allow alert in production code
  - <script type="text/javascript">
  - alert("Hello Java Script");
  - <script>
- Better solution would be to
  - <script type="text/javascript">
  - Var msg = "Hello JavaScript";
  - Console.log(msg);
  - <script>

JavaScript

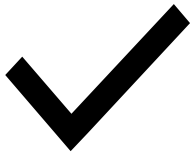
is

case

SENSITIVE

Case Sensitive & White space insensitive

REVATURE

- Alert(“Hello World!”); 
- alert(“Hello World!”); 
- alert(“Hello World!”); alert(“Hello World again!”);
  - Not a good practise
- alert(“Hello World!”);
- alert(

  “Hello World!”

  );

Structures

REVATURE

- Forgive omissions – Not a good practice. Eventually, you will run in to problems
- Execution order is sequential statements one after another
  - Example, move your script tag from head to body
- //this is a line comment
- console.log(“hello!!”); //comments can be written here
- /* this part of
  the code will not get
  executed
  because this is block comment
  */

Internal & External

REVATURE

- HTML4
  - <script src=myscript.js type="text/javascript">
  - </script>
- HTML5
  - <script src=myscript.js>
  - </script>
  - Code editors might complain/add them when missed

Get the Element with specific ID

REVATURE

- Pre-requisite: Create a HTML page with tag containing value for ID attribute
- Problem: Assuming the below section is an example code, how to change the value of element using the ID
  - <section>
    - <div id=*"results" class="border"*>
    - Example of results value would be displayed here
    -
  - </section>
- Answer: document.getElementById(“results");
- Solution:
  - <script type=*"text/javascript"*>
  - var resultsDiv = document.getElementById("results");
  - resultsDiv.innerHTML = " This message is displayed by JavaScript "
  - </script>

Problem

REVATURE

- Create a new JS file and move the content of <script> tag into index.js file
  - var msg = "Hello JavaScript";
  - console.log(msg);
  - var resultsDiv = document.getElementById("results");
  - resultsDiv.innerHTML = " This message is displayed by JavaScript "
- Include the JS into HTML by using src attribute
  - <script type=*"text/javascript" src="index.js"*> </script>
  - If the index.js is not in the same directory please use appropriate relative path

Script tag

REVATURE

- Script tag can not have a self closing tag. It should mandatorily have ending tag
- Script tag can be included head or body.
- However, better practice would be to include in body at the very end
- If there are multiple script tags the execution order of including them is important

Creating Variables

REVATURE

- var myVariable;
- var myVariable111;
- var 111myVariable; ✗
- var emailAddress;
- var todaysDate;
- var currentFiscalYear;
- var x;
- x = 100;
- var x = 100, y = “Hello”, z = “Hola’; ✗
- var i = “I’d like to “eat” all the time”; ✗
- var i = “I’d like to \“eat\” all the time”;

Variables and Types

REVATURE

- //Assign the variables with the Assignment Operator, the =
- a = 10; //number
- b = "10"; //String
- c = true; //boolean
- d = {}; //object
- e = null; //null
- //f //undefined
- g = (0/0); //NaN
- h = []; //Array
- i = myName;

Variables and Types

REVATURE

- Number, String, Boolean, Array, Object, Undefined
  - var msg = "Hello JavaScript";
  - console.log("msg is " + typeof(msg));
  - var resultsDiv = document.getElementById("results");
  - console.log("resultsDiv is " + typeof(resultsDiv));
  - var none;
  - console.log("none is " + typeof(none));
  - var aNum=10;
  - console.log("aNum is " + typeof(aNum));
- Variables can be assigned values without declaration
  - nonExistentVariable = "This shouldn't work";
- Force declaration of variables using “use strict”

Practice on Console

REVATURE

- var foo = 5, foo1 = 6, foo2;
- var bar = “5”; bar1 = 6, bar2 = “world”;
- console.log(foo + bar);
- console.log(foo1 + bar1);
- console.log(foo2 + bar2);
- var myString = 'hello';
- var newString = myString + ‘ world';

Operators

REVATURE

- Arithmetic + - / *
- Assignment =
- += -= /= *= (+= or =+)
  - a = b + c;
  - a = a + 10;
  - a += 10;
- Operator precedence * / then + -
- = Assignment, == Equality, === Strict Equality
- <, >, =, ==, ===, !=, !==

Operators

REVATURE

- Logical operators && ||
- Modulus %
  - remainder = year % 4 (2003 % 4)
- Increment
  - a = a + 1; a+= 1; a++; ++a;
- Decrement
  - a = a – 1; a-= 1; a--; --a; (also called Unary ++, --)
- Ternary
  - <condition> ? <true> : <false>

Examples

REVATURE

- If (a > 20){
  - //code goes here;
- }
- var x = 100;
- if (x< 90){
  - console.log("x is less than 90");
- } else{
  - console.log("x is greater than 90");
- }

Examples

REVATURE

- var playerOne = 100;
- var playerTwo = 120;
- if (playerOne > playerTwo) {
  - console.log(“high scorer is playerOne”);
- } else{
  - console.log(“high scorer is playerTwo”);
- }
- var highScore = (playerOne>playerTwo) ? playerOne : playerTwo

Conditional statements

REVATURE

- If, If else if, If else
  - if(none==undefined){
  - console.log("none is undefined");
  - }
  - if(aNum ="10"){
  - console.log("10 is 10");
  - }
  - if(aNum =="10"){
  - console.log("10 is 10");
  - }
  - if(aNum ==="10"){
  - console.log("10 is 10");
  - } else{
  - console.log("10 is not 10");
  - }

== vs ===

REVATURE

- if (a==b){
- console.log("Yes, they're equal");
- } else {
- console.log("No, they're not");
- }
- if (a===b){
- console.log("Yes, they're equal");
- } else {
- console.log("No, they're not");
- }

True and False

REVATURE

- //true and false
  - 5 == 5 //true
  - 5 != 6 //true
  - 5 == "5" //true
  - 5 === "5" //false

- 5 == "5" //true because it performs Type Coercion
- 5 === "5"//Does not perform Type Coercion,
  - //checks the data type first if != then false

Truthy & Falsy

REVATURE

- false, 0 (zero), “”, null, undefined, NaN
- Falsy values false, 0 and “” are equivalent
  - var a = (false == 0); // true
  - var b = (false == ""); // true
  - var c = (0 == ""); // true
- null and undefined are not equivalent to anything except for themselves
  - var d = (null == false); // false
  - var e = (null == null); // true
  - var f = (undefined == undefined); // true
  - var g = (undefined == null); // true

Truthy and Falsy

REVATURE

- Falsy value NaN is not equivalent to anything including NaN
  - var j = (NaN == null); // false
  - var k = (NaN == NaN); // false
  - var a = !!(0); // variable is set to false
  - var b = !!("0"); // true

```
var x = [];
if(x.length && 55){
        console.log('I am True');
}else if( x < 55){
        console.log('x is less than 55');
}else{
        console.log('I am not true');
}
```

Loop Statements

REVATURE

- statement one
- statement two

- statement one
- statement two

- statement one
- statement two

- statement one
- statement two

- repeat 5 times {
  - statement one
  - statement two
- }

- Repeat 1000 times
- For every link
- When to stop the loop?

While Loop

REVATURE

- var a = 1;
- if (a < 10) {
  - console.log(a);
- }

- var a = 1;
- while (a < 10) {
  - console.log(a);
  - a++;
- }

- var a = 1;
- while (a < 10) {
  - console.log(a);
- } **Indefinite loop**

- var a = 1;
- do {
  - console.log(a);
  - a++;
- } while (a < 10);

Setup index, check the condition, increment the index

For Loop

REVATURE

- for (a = 1; a < 10; a++) {
  - console.log(a);
- }

- for (a = 1; a < 10; a++) {
  - console.log(a);
  - if (a == 5) {
    - break; //break jumps out of the loop
  - }
- }
- //unlike break, “continue” skips the iteration not the entire loop

Functions

REVATURE

- Functions Can Be Used as Values
- Function declaration
  - var f1 = function (x, y) {return x * y};
- Functions using constructor
  - var myFunction = new Function(“x", “y", "return x * y");
  - var x = myFunction(5, 10);
- Function Hoisting – Call the function before creating
- Self-Invoking functions (function () {
  var x = "Hello World!";
  })();

Functions

REVATURE

- Reusable, modular statements
  - showMsg("information to be displayed");
- Functions will not execute unless they are called (except self-invoking)
- Functions can accept information – Inputs
- Also can return values - Outputs

```
function showMsg(){
      console.log(“Hello World!!”);
}

function myFunction (x, y) {
            var myVar = x * y;
            console.log(myVar);
            return myVar;
}

myFunction (10, 20);
var result = myFunction(135, 654);

myFunction (100, 200, 300); //300 will be ignored
myFunction (100) // second parameter will be passed with a value undefined
```

Functions

REVATURE

```
function myName(){
        //print to the console
        console.log(“Yuvi");
        return 55;
}

var myString = 'hello';
function changeString(x){
        var newString = x + 'world';
        console.log(newStrin
g);
}
```

```
function returnChangedString(x){
        console.log(x);
        x = x + 'world';
        return x; //Is the variable that was past into x changed?
}

function returnChangedNumber(y){
        console.log(y);
        y = y + 10;
        return y;
}
```

String functions

REVATURE

- String functions
  - length
  - indexOf
  - lastIndexOf
  - search
  - slice (start, end)
  - substring (start, end)
  - substr (start, length)
  - replace
  - toUpperCase
  - toLowerCase
  - concat
  - charAt(position)
  - charCodeAt(position)

Pop Quiz

Variable Scope

REVATURE

```
function simpleFunction() {
    // lots of code
    var foo = 500;
    // lots of code
    console.log(foo);  500
}

simpleFunction();
console.log(foo);  undefined
```

- Local variables aren't accessible outside of their scope

Variable Scope

REVATURE

```
var foo;

function simpleFunction() {
    // lots of code
    foo = 500;
    // lots of code
    console.log(foo);  500
}

simpleFunction();
console.log(foo);  500
```

- Global variables are accessible inside and outside of the functions

Scopes

REVATURE

```
var testGlobal = true;

function testScopes(){
    console.log("testGlobal is : " + testGlobal);

    var testLocal = true;
    console.log("testLocal is : " + testLocal);
}

testScopes();
//console.log("testLocal outside of function is : " + testLocal);
```

- Global and Local Scopes
  - Variables declared inside functions have local scope
  - Variables declared outside functions have global scope
  - Global variables go out of scope when you close the page
- Show every function that is shared
- Scope changes inside a function

Objects

REVATURE

- JavaScript objects are mutable
- Objects are collection of name value pairs
- Methods are actions that can be performed on objects
- Objects combine the data with functions and give it a name
- Create object in different ways – literal, new, constructor
- Associate a method with an object using objectName.methodName = functionName;

Create Objects

REVATURE

- var playerName = “Joe Montana”;
- var playerScore = 600;
- var playerRank = 2;

variables

- var player = new Object();
- player.name = “Joe Montana”;
- player.score = 600;
- player.rank = 2;

properties

Literal

REVATURE

- var player1 = {name: “Joe Montana”, score : 600, rank : 2};
- var player2 = {name: “Tom Brady”, score : 650, rank : 1};
- fuction playerDetails(){
  - console.log(this.name+ “ has a score of “ +this.score);
- }
- player1.logDetails = playerDetails;
- player2.logDetails = playerDetails;

- player1.logDetails();

Constructor

REVATURE

- function playerDetails(name, score, rank){
  - this.name = name;
  - this.score= score;
  - this.rank= rank;
- }
- var player1 = new playerDetails(“Tom”, 650, 1);
- console.log(player1.name + "   " + player1.score);

Arrays

REVATURE

- var singleValue = 100;
- var multipleValues = [];
- Arrays have zero based index
  - multipleValues[0] = 10;
  - multipleValues[1] = 20;
  - multipleValues[2] = 30;
  - multipleValues[3] = “Hello”;
- multipleValues = [10, 20, 30, “Hello”]
- var x = “This is an sample string”;
- var words = x.split();

Arrays

REVATURE

- someFunction(params); //to call a function

- **//methods are functions that belong to an object**
- someObject.someMethod(); //to call a method

- var multipleValues = [10,20,30,40,50];
- var reversedValues = multipleValues.reverse();
  .sort();
  .join();

- console.log(reversedValues.join()); //50,40,30,20,10

Arrays

REVATURE

- Array containing an objects
  - var results = [{
  - name : "jQuery", language : "JavaScript", score : 4.5,
  - showLog : function() { console.log("showLog function");},
  - owner : {login : "yuvi",userid : 12345}
  - }, {
  - name : "jQuery2", language : "JavaScript", score : 3.5,
  - showLog : function() { console.log("showLog function");},
  - owner : {login : "yuvi",userid : 4567}
  - },
  -];

More Examples

REVATURE

- Variable Declaration
  - var foo = 3;
  - foo = “three”; or foo = true;
  - var foo = new Array(1, 2, 3); or var foo = [1, 2 , 3];
  - var obj = new Object(); or var obj = {};
- Function Declaration
  - Function foo () {return “Hello World”;}
  - Var foo = Function () {return “Hello World”;}

JSON Example

REVATURE

```
var book = {
                title: "Harry Potter",
                year: 2001,
                author: {
                        name: "JK",
                        dob: 1965
                }

};
```

Loops – JSON Examples

REVATURE

- For and While loop examples
  - var person = {fname:“Jim", lname:“Carter", age:16};
    var text = “”, x;
    for (x in person) {
        text += person[x];
    }
  - var cars = [“Honda", “Camry", “Fusion", “Mazda"];
    var i = 0;
    var text = "";

    while (cars[i]) {
        text += cars[i] + "
";
        i++;
    }

DOM

REVATURE

- Document – Both HTML & browser view
- Object – Elements, tags of the document
- Model – Agreed terminology

Document Object Model (DOM)

web page pieces agreed-upon set of terms

JavaScript Examples

REVATURE

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
alert("Hello! I am an alert box!");
}
</script>
</head>
<body>
<input type="button" onclick="show_alert()" value="Show
   alert box" />
</body>
</html>
```

JavaScript Examples

REVATURE

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE=”JavaScript”>
// JavaScript statements go here
function processOrder() {
// More JavaScript statements go here
}</SCRIPT>
</HEAD>
<BODY>
<FORM NAME=”myForm”>
<INPUT TYPE=”button” NAME=”processOrder” VALUE=”Click to
   process your order” onClick=”processOrder();”>
</HTML>
```

JavaScript Examples Cont.

REVATURE

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>JS Demo</title>
    <script type="text/javascript">

        window.addEventListener("load", function () {
            var bt = document.getElementById("btnGo");
            bt.onclick = test;
        }, false);

        function test() {
            var times = tbInput.value;
            var place = document.getElementById("test");
            place.innerHTML = "";
            for (var i = 0; i < times; i++) {
                place.innerHTML += "<p> " + (i + 1) + " </p>";
            }
        }
    </script>
</head>
<body>
    <div>
        <input type="text" id="tbInput" />
        <input type="button" id="btnGo" value="Make Paragraphs" />
        <div id="test">
        </div>
    </div>
</body>
</html>
```

Check it out and post the results !!!

REVATURE

- var myImage = document.getElementById("dogs");
- myImage.onclick = function() {
- 	console.log("you clicked on the dogs image");
- };
- This will work as long as the script tag is at the end of body
- Move the script tag to head
- Move the above lines into a function and call the function in anonymous function for window.onload

Closures

REVATURE

```
// JavaScript
var x = 1;

function someFunction() {
  // Works as it wraps 'x' with a closure
  var y = x;
}

// Much Later
someFunction();
```

- Allows access to outer variables within inner scopes
  - Regardless of lifetime

Asynchronous function call

REVATURE

- Define a function with two parameters. Second parameter is a function
  - function showItThen(msg, hello){
  - showIt(msg);
  - hello();
  - }
- Call the function like below and pass the second value as anonymous function
  - showItThen("showItThen called ", function(){
  - console.log("hello function is called");
  - })
- While passing the value to an object assume the data type as callable function
- These are similar to Lambda in Java

Common Mistakes

REVATURE

- Calling a non-existing function due to a typo
- Typos in DOM methods are common
- Using a non-existent object method
- Assignment instead of equality = instead of ==
- Missing parameters in function calls lead to unexpected results

Avoid these

REVATURE

- document.write
  - Its great if you are using this onload. But while manipulating the DOM elements this wipes out the content & replaces the entire DOM
- Browser sniffing
  - Old code was not compatible in all browsers. Hence detecting the browsers were necessary. Not anymore,
- Eval
  - var a = “alert(‘”, b=“hello”, c=“’);” eval(a + b + c);
  - This can actually inject code
- Psuedo protocols
  - <a href=“javascript:myfunction()” />
  - <a href=“none.html” onclick=“myfunction(); return false” />