Javascript

Life of the web

Aprameyah
Technologies Pvt. Ltd.

About Me

- Maruthi R Janardhan
  - Been doing java since jdk 1.2
  - Been with IBM, ANZ, HCL-HP, my own startup Leviossa..
  - Total 16 years programming C, C++, Java, Javascript, Ruby, Perl, Python, PHP, etc

What’s Javascript

- JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages. With the chrome engine - also a server side language.
- Less server interaction
- Immediate feedback to the visitors
- Increased interactivity
- Richer interfaces

Ecmascript

- Javascript originated in Netscape [1996]. Microsoft tried its own variant calling it JScript

- End 1996 Netscape submitted Javascript to Ecma International for standardisation.

- June 1997 - First version of ecmascript

- Dec 2011 - Version 5.1 of ecmascript also known as ES5

    - Google V8 engine supports ES 5.0

Its a Script!

- First and foremost - its a script. Code does not need to be in functions or classes. Variables and code can be in global space. No main() method

- No compiler - its an interpreter. You will not see syntax issues until you run into that line of code during execution

  - Code coverage testing is really important

- Memory management is via a garbage collector

No Pointers

- No pointers, however function references do exist
- Variables cannot be dereferenced
- Parameter passing is by reference
- Array name is not a pointer to its start and no pointer arithmetic and no running off the edge

```
var k = [];
k[3]=5;
console.log(k.length);//4
console.log(k);//[ , , , 5 ]
```

Basics

- Javascript is a duck-typed language. If it looks like a duck and it quacks like a duck, it is a duck.
    - Variables don’t need type declaration
- Certain implicit objects exist
- JS is case sensitive
- All statements end with ; (though not needed)

```
<script type="text/javascript">
     var i=10;
     console.log(“Hello World! "+i);
</script>
```

- Functions:

```
function doSomething(data){
     console.log(“Hello World! "+data);
}
```

Basic Structures

```
for(var i=0;i<10;i++){

}
```

```
for (x in arr) {
    console.log(x);
}
while (i < 10) {

}
do {

}
while (i < 10);
```

```
switch (new Date().getDay()) {
case 0:
    console.log("Sunday");
    break;
case 6:
    console.log("Saturday");
    break;
}
```

```
typeof "John"  // string
typeof 3.14   // number
typeof false // boolean
typeof [1,2,3,4]    // object
typeof {name:'John', age:34} //
object
```

Basics

- You can include scripts in other files: <script src="myscript.js"></script>
- Or using the require("./myscript.js") on server side
- Operators:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Logical Operators

- == equal to
- === equal value and equal type
- != not equal
- !== not equal value or not equal type
- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to
- && and
- || OR
- ! Not

Type based comparison

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
===	equal value and equal type	x === "5"	false
		x === 5	true
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== "5"	true
		x !== 5	false

- false, “”, NaN, undefined, null are all treated as false in a conditional statement

- var x=5; x==“5” will be true but x===“5” will be false

Bitwise Operators

- Bitwise operators work on 32 bit numbers
- & AND
- | OR
- ~ NOT
- ^ XOR
- << Left Shift
- >> Right Shift

JS Display

- Five ways to display from JS
  - Writing into an alert box, using window.alert().
  - Writing into the HTML output using document.write().
  - Writing into an HTML element, using innerHTML.
  - Writing into the browser or node console, using console.log(). [This is the only type applicable to server applications]
  - Changing CSS styles

JS Datatypes

- String, Number, Boolean, Object, Array, Undefined
- typeof operator can be used to find the type of a variable
- Redeclaring a variable does not lose its value
- Assigning to a var without decl will make it global
- Declaring arrays and objects (JSON):

```
var people = ["Mahesh", "Ganesh", "Srini"];            // Array
var person = {firstName:"Rahul", lastName:"Dravid"};  // Object
```

Type Conversions

```
console.log(typeof String(123)); //Copy constructor kind of casting
console.log(typeof Number(“5.6”));

5 + null    // 5
"5" + null  //"5null"
"5" + 1     //"51"
"5" - 1     //4
```

There is no cast in javascript

Arrays

```
//Create array (Avoid the new Array())
var people = ["Will", "Kelly", “Trevor"];

//Access with index
people[0] = "Jerry";

alert(typeof people);//Object

//Add elements with index
people[people.length] = “Mark";
//Same as
people.push("Mark");

//Associative arrays are objects
people["Kelly"] = "Willow";
alert(people.length); // 0
alert(people[0]); // Undefined

for (person in people){}
```

Functions

- Functions are defined like below and can have return values.

```
function makeDirectory(parent,newDir){
        //Whatever code here
        return "someval";
}
```

- Just like variables have no type functions have no return type
- No function overloading in javascript
- Parameters dont even have to be defined. We can call a function with as many arguments as we like. If we pass less, the remaining are undefined
- Every function gets a local array called “arguments”

Function Parameter Passing

```
function changeStuff(num, obj1, obj2)
{
  num = num * 10;
  obj1.item = "changed";
  obj2 = {item: "changed"};
}

var num = 10;
var obj1 = {item: "unchanged"};
var obj2 = {item: "unchanged"};

changeStuff(num, obj1, obj2);

console.log(num);
console.log(obj1.item);
console.log(obj2.item);
```

Function Type

- Function is a datatype. So function references can be stored in variables. Functions can be anonymous too… and it can get crazy

```
function doSomething(){
   console.log("hello world!");
   console.log(doSomething.x);
}
console.log(typeof doSomething);
var fnVar = doSomething;
fnVar.x = 10;
fnVar();
fnVar.run = function(){
   console.log("running...");
};
doSomething.run();
```

Hoisting

```
function test() {
    console.log(a);
    console.log(foo());

    var a = 1;
    function foo() {
        return 2;
    }
}

test();
```

Lets Try It

- Write a function to reverse an array thats passed in. The array could be passed as an array variable or as a comma separated list of parameters

Lets Try It

- Write a function that computes the sin of a number and notifies another function with the computed value

Threading

- Javascript programs are single threaded!!!!

- Threads can be partly simulated using a timeout event

```
setTimeout(function,delay);
```

- Convert the previous sin computation logic to be asynchronous with the help of setTimeout

Objects

- Objects are defined using JSON
- There is no class definition. We can add fields and methods to objects on the fly

```
person.run = function(){
    alert("Person running");
}
person.run();
```

- Object properties can be accessed with variables too!

```
person["lastName"] = "Munjal";
```

“this” context

- The context of a function, what is referred with the this keyword, in JavaScript depends on how a function is invoked, not how it’s defined.

```
var fullname = 'Kelly Perry';
var obj = {
    fullname: 'Will Berger',
    prop: {
        fullname: 'Shahid Khan',
        getFullname: function() {
            return this.fullname;
        }
    }
};

console.log(obj.prop.getFullname());
var test = obj.prop.getFullname;
console.log(test());
```

Call & Apply

- Call and Apply lets us set the “this” context for a function. Infact functions need not be part of objects to refer to the “this” context

```
console.log(test.call(obj.prop));

function concatAndGetLength(str){
   return (this+str).length;
}
console.log(concatAndGetLength.call(“Hello","world"));

console.log(concatAndGetLength.apply("Hello",["world"]));
```

Prototypes

- Each object has an internal link to another object called its prototype. That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype.

Prototypes

```
var c = new Car(10,"Red");
var d = new Car(20,"Blue");
d.start = function(){
    console.log("Starting car d");
};

d.start();
//c.start();//Invalid

Car.prototype.start = function(){
    console.log("starting the car");
};

c.start();
d.start();//The object function overrides the prototype function
```

Defining Functions For Classes

```
function MyObject(name, message)
{
    this.name = name.toString();
    this.message =
message.toString();
    this.getName = function() {
        return this.name;
    };

    this.getMessage = function() {
        return this.message;
    };
}
```

```
function MyObject(name, message)
{
    this.name = name.toString();
    this.message =
message.toString();
}
MyObject.prototype.getName =
function() {
    return this.name;
};
MyObject.prototype.getMessage =
function() {
    return this.message;
};
```

Custom Classes

- JavaScript is a prototype-based language which contains no class statement
  - uses functions as classes.
  - Defining a class is just like defining a function
  - The function is treated as constructor

```
var Person = function () {
    console.log("Person constructor...")
};
var person1 = new Person();
```

Defining a class

- this or a special property “prototype” is used to add properties and methods to the class

```
var Person = function () {
    console.log("Person constructor...")
    this.firstName = "";
    this.lastName = "";
    this.getName = function(){
        return this.firstName+this.lastName;
    }
};

Person.prototype.getFullName = function(){
    return this.firstName+this.lastName;
}
var person1 = new Person();
person1.firstName = "Kelly";
person1.lastName = "Perry";
console.log("Full name: "+person1.getName());
console.log("Full name prototype: "+person1.getFullName());
```

Extending Classes

- Object.create - creates an object instance using a prototype without invoking the constructor. Hence defining methods on prototype is better than doing in constructor

```
// Define the Student constructor
function Student(firstName, subject) {
  Person.call(this, firstName);
  // Initialize our Student-specific properties
  this.subject = subject;
};

Student.prototype = Object.create(Person.prototype);

// Set the "constructor" property to refer to Student
Student.prototype.constructor = Student;

//Override methods
Student.prototype.getFullName = function(){
   return this.firstName+this.lastName+"("+this.subject+")";
};
```

Lets try it

- Create an array of Cars with properties (Brand, model, year, fueltype). Then extend SportsCar from car (Sports car has racingTeam property) and add to array. Print the array.

A Collection Implementation

- Create a collection class and a set class extending from collection. Sets do not allow adding duplicates. Required methods:
  - add(obj)
  - get(index)
  - size()
  - remove(index)

Strings

- Strings are a type but internally also handled as an object. Has methods and properties

- .length

- indexOf(), replace(), substr()

Numbers

- Numbers are Always 64-bit Floating Point
- Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.
- NaN is a JavaScript reserved word indicating that a value is not a number.
- Also have properties and methods:
    - MAX_VALUE, MIN_VALUE, NEGATIVE_INFINITY, NaN, POSITIVE_INFINITY
    - toString(), toFixed(), toPrecision(). Also there is a Math class
- Try not to use these.. JS is not for complex biz logic anyway.

RegEx

```
var emailPattern = /[^\s]*@.*\.com/i;

var str = "My email address is: mark@abc.com";
var result = str.replace(emailPattern,"<EMAIL>");
console.log(result);


console.log(emailPattern.test("wonder if we have an email address here
maruthi@le.com!"));
```

Lets try it

- Eliminate email addresses from an array of text fields (Use string regex replace)

Delete operator

- delete operator can be used to delete properties of an object

```
var student = {
    name : "Mahesh Singh",
    sub: "Math",
    rollno : 12
};
console.log(student);
delete student.rollno;
console.log(student);
```

HasOwnProperty

- hasOwnProperty(key) - is a way to check if the object has a property

Lets Try It

- Loop through the properties of an object with for(x in obj){}

Namespaces

- Just like packages in java, we need a way to confine changes to our app since we end up modifying exiting objects

- There is no VM running for long… any impact is anyway limited to a page. Still why protect?

- Namespaces are just plain object in JS. Check and create a namespace and sub namespace

```
var MYAPP = MYAPP || {};
MYAPP.core = MYAPP.core || {};
```

Lets Try It

- Move the Collection and Set classes into MyApp namespace

Closures

- Closure is a way of implementing private class state in javascript by playing on variable scope

```
function add() {
    var counter=0;
    counter += 1;
    console.log(counter);
}

add();
add();
------------------------------
var add = (function () {
    var counter = 0;
    return function () {
        counter += 1;
        console.log(counter);
        return counter;
    };
})();

add();
add();
```

Closures As Function Factories

```
function notifyUser(email) {
    return function(text) {
        console.log("Sending '"+text+"' to "+email);
    };
}

var notifyWill = notifyUser('wberger@lo.com');
var notifyKelly = notifyUser('kperry@lo.com');

notifyWill("Hello, you have a new join request");
notifyKelly("Hello you are late to work!");
```

Closures As Modules

```
var counter = (function() {
    var privateCounter = 0;
    function changeBy(val) {
        privateCounter += val;
    }
    return {
        increment : function() {
            changeBy(1);
        },
        decrement : function() {
            changeBy(-1);
        },
        value : function() {
            return privateCounter;
        }
    };
})();

console.log(counter.value());
counter.increment();
counter.increment();
console.log(counter.value());
counter.decrement();
console.log(counter.value());
```

Singletons With Closures

```
var mySingleton = (function() {
    var instance;
    function init() {
        function privateMethod() {
            console.log("I am private");
        }
        var privateVariable = "Im also private";
        var privateRandomNumber = Math.random();
        return {
            publicMethod : function() {
                console.log("The public can see me!");
            },
            publicProperty : "I am also public",
            getRandomNumber : function() {
                return privateRandomNumber;
            }
        };
    };

    return {
        getInstance : function() {
            if (!instance) {
                instance = init();
            }
            return instance;
        }
    };
})();
```

Closures & Objects

```
var cars = [{brand: 'Fiat', model:'Punto'},{brand: 'Hyundai',
model:'Santro'},{brand: 'Maruti', model:'Swift'}];
for (var i = 0; i < cars.length; i++) {
    cars[i].printLocation= function() {
        console.log(this.model+" is in location: "+i);
    };
}

cars[0].printLocation();
```

Fix the looping problem

```
for (var i = 0; i < cars.length; i++) {
   cars[i].printLocation = function(i) {
      return function() {
         console.log(this.model + " is in location: " + i);
      };
   }(i);
}

cars[0].printLocation();
```

Lets Try It

- Make the collection class we wrote earlier as a closure to encapsulate the array

Lets Try It

- Create a dialog class that supports ok/cancel buttons.

- Provide a subclass that also adds an ignore button