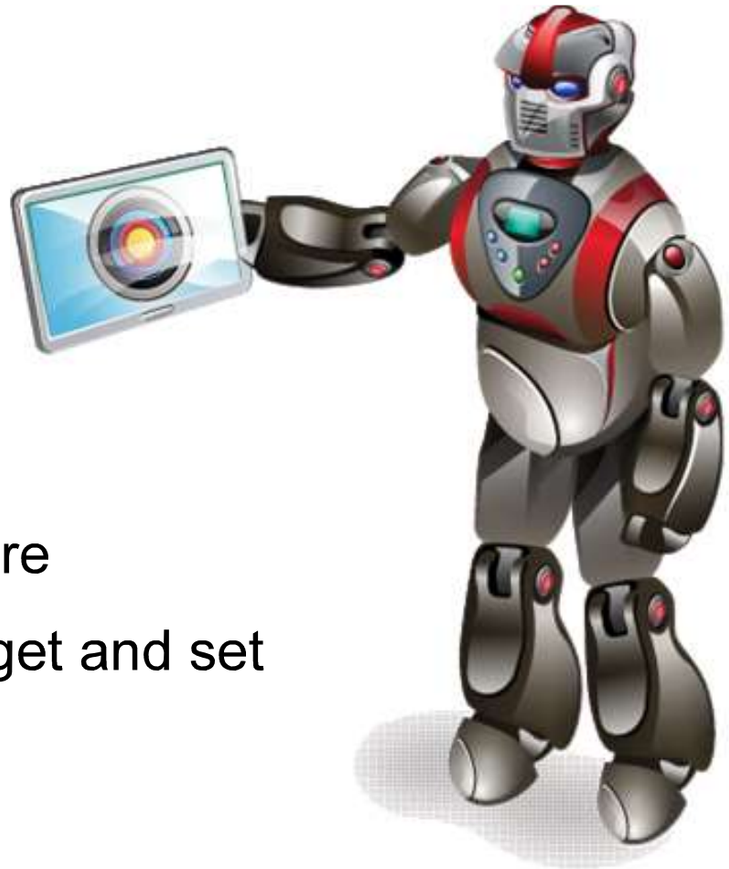
3

JavaScript Fundamentals

Objectives

After completing this lesson, you should be able to:

- Write JavaScript code to:
  - Declare variables
  - Create an object
  - Create and call a function
  - Create and iterate arrays
- Write JavaScript arrays to store data
- Define JavaScript objects as a key-value store
- Access the properties of an object by using get and set



Topics

Part I:

- The JavaScript Grammar
- Variables and Types
- Expressions and Operators
- Statements and Loops

Part II:

- Objects
- Arrays
- Functions

Comparing JavaScript and Java

JavaScript:

```
var x = 5;
var car = Object.create(Car.prototype);
var hello = function myFunction() {
    console.log("Hello world!");
}
```

Java:

```
int x = 5;
Car car = new Car();
public void myFunction() {
    System.out.println("Hello world!");
}
```

Examining the Grammar of JavaScript

- JavaScript is case-sensitive:

```
getElementById() ! = getElementByID()
```

- Escape s equence: `\u`

Example:

Line feed

```
var x = 15; // \u000A is allowed in comments

var a = "This line wouldn't be accepted in Java\u000A";
```

Examining the Grammar of JavaScript

- Comments:

```
// This is a single-line comment, until the end of line
/* And here is a multi-line
   comment, notice that this
   type of comments cannot be nested */
```

- Literals:

```
null
"True"
'true'
true
15.5
```

Automatic Semicolon Insertion

```
var x = 15.5
var y = "Total";
```

semicolon is inserted

```
a = b + c
(d + e) = f;
```

semicolon is NOT inserted

Examining the Grammar of JavaScript

- Identifiers

```
this_is_a_variable_name
_another_variable_name
$a_third_variable_name
```

- Reserved Words

break	do	instanceof	typeof	case
else	new	var	catch	finally
return	void	continue	for	switch
while	debugger	function	this	with
default	if	throw	delete	in
try				

Example with *strict* mode:

```
function findPrice() {
 'use strict';
 price = 1200;   // this line causes an uncaught
                 // ReferenceError
}
```

***Non-strict* mode or normal JavaScript:**

```
function findPrice() {
 price = 1200;    // this line does not cause an error
}
```

Topics

Part I:

- The JavaScript Grammar
- Variables and Types
- Expressions and Operators
- Statements and Loops

Part II:

- Objects
- Arrays
- Functions

Types

Type	Example
Undefined	var a = undefined;
Null	var b = null;
Boolean	var d = true;
String	var c = "This is a string";
Number	var n = 15.5;
Object	var o = Object.create(Object.prototype);

Declaring a Variable

Variable declaration:

```
var a;
var x = 15.5;
var y = "Total";
```

Variable in a function:

```
function declareVariable() {
    if (true) {
      var x = "Declared in a block of code";
    }
   console.log("Value of x: " + x);
};
declareVariable();
```

Topics

Part I:

- The JavaScript Grammar
- Variables and Types
- Expressions and Operators
- Statements and Loops

Part II:

- Objects
- Arrays
- Functions

Creating an Array and an Object

- Initializers:

```
var courses = ["Java","JavaScript","Web Services"];
var students = [12,5,7];
```

```
var course = {name: "JavaScript", students: 5, frequency: "daily"};
```

- Creating an array with `new`:

```
var students_new = new Array();
var students_new_init = new Array(12,5,7);
```

- Creating an object with `new`:

```
var courses_new = new Courses();
var courses_new_init = new Courses(12,5,7);
```

Accessing an Array and an Object

```
var myObject = {
     name: "Ring",
     diameter: 12,
     specs: {
       material: "Wood",
       waterProof: false
     }
};
var myArray = ["one", "two", 3, "four", 5];
```

- By using the dot notation (only for objects):

```
myObject.diameter;          ← 12
myObject.specs.waterProof;  ← false
```

- By using the bracket notation:

```
myObject["diameter"];           ← 12
myObject["specs"]["material"];  ← "Wood"

myArray[0];  ← "one"
myArray[5];  ← undefined
```

Creating and Calling a Function

- Function definition:

```
function multiply(x, y) {
    var result = x * y;
    return result;
}
```

- Function invocation:

```
var total = multiply(135,6);
```

Working with Operators

Arithmetic	Relational	Logical	Assignment	Special
++	==	&&	=	,
--	!=	||	**Bitwise**	delete
+	===	!	&	in
-	!==	**Conditional**	|	instanceof
*	>	? :	~	void
/	>=		^	typeof
%	<		>>	
	<=		<<	

Operators

- Arithmetic operators:

```
9 - 5;          ←  4
[9] * ['5'];    ←  45
"9" + 5;        ←  A String :"95"
```

- Relational operators:

```
9 == '9';       ←  true
9 === '9';      ←  false
```

- Logical operators:

```
var a = "string" && "another";  ←  "another"
var o = "string" || "another";  ←  "string"
var n = !"string";              ←  false
```

Operators

- Assignment operator:

```
a = b;
x += 1;
```

- Bitwise:

```
5 & 6;          // 101 & 110 = 100
```

4

- Conditional operator:

```
condition ? expression_1 : expression_2

var status = (age >= 18)? "major" : "minor";
```

Special Operators

- Comma operator:

```
for (var i = 0, j = 9; i <= 9; i++, j--)
   console.log(i + " , " + j);
```

- `delete` operator:

```
var i = { a : 5, s : "String", b : 15 };
delete i.s;      ← returns true. i = {a : 5, b : 15}
"b" in i;        ← true
"s" in i;        ← false
```

```
var i = [ 5, "String", 15 ];
delete i[1];     ← returns true. i = [5, undefined, 15]
0 in i;          ← true
1 in i;          ← false
```

Special Operators

- `void` operator:

```
<a href="javascript:void(0);">This link does nothing</a>
<a href="javascript:void(aMethod());">This link calls
    aMethod</a>
```

- `typeof` operator:

```
var a = [1, 2, 3, 4];
```

```
typeof undefined;      "undefined"
typeof "Hello";        "string"
typeof a;              "object"
typeof a[0];           "number"
```

Topics

Part I:

- The JavaScript Grammar
- Variables and Types
- Expressions and Operators
- Statements and Loops

Part II:

- Objects
- Arrays
- Functions

Writing Statements

- **A block of code:**

```
{
    <statement 1>
    ...
   <statement n>
}
```

- **Declaring a variable:**

```
var x = 15;
```

- **Conditional:**

```
if (expression) {
    <statement 1>
} [else { <statement 2> }]
```

- **switch and break:**

```
switch (expression) {
    case label_1:
      <statements 1>
      [break;]
    case label_2:
      <statements 2>
      [break;]
    ...
    default:
      <statements n>
      [break;]
}
```

Writing Statements

- **return:**

```
function multiply(x, y) {
    var result = x * y;
    return result;
}
```

- **with:**

```
with (object) {
    <statements calling the properties of the object>
}
```

- **try and throw:**

```
function myFunction() {
    try {
      functionCanThrowAnException();
      if (condition) {
          throw "myException";
      }
    }
    catch (e){
      logError(e);
    }
    finally {
      closeFiles();
    }
}
```

Statements

- Labeled statement:

```
myLabel:
    <statement>
```

- debugger:

```
function iNeedToStopHere(){
    debugger;
   // do debugging stuff
}
```

- Continue:

```
for (var i=0; i<10; i++) {
    if (i === 5) continue;
      console.log("This number is:" + i);
}
```

While Loop

- **while:**

```
var i = 0;
while (i < 10) {
    console.log(i+=1);
}
```

- **do/while:**

```
var i = 0;
do {
    console.log(i+=1);
} while (i < 10);
```

semicolon

for Loop

- **for:**

```
for (var i = 1; i <= 10; i+=1){
    console.log(i);
}
```

- **for/in:**

```
var myObject = {
    one: "value of property one",
    two: "value of property two"
}
```

```
for (var myVar in myObject) {
    if (myObject.hasOwnProperty(myVar)) {
      console.log(myVar);
    }
}
```

Topics

Part I:

- The JavaScript Grammar
- Variables and Types
- Expressions and Operators
- Statements and Loops

Part II:

- Objects
- Arrays
- Functions

Creating Objects

- **Creating an empty object:**

```
var obj_1 = {};
var obj_2 = new Object();
var obj_3 = Object.create(null);
```

- **Creating an object:**

```
var person = {
    "full-name" : "John Doe",
     age: 35,
     address: {
       address_line1: "Clear Trace, Glaslyn, Arkansas",
       "postal code": "76588-89"
     }
};
```

Creating Objects

- **Creating an object with `new`:**

```
function Tree(type1, height1, age1) {
    this.type = type1;
    this.height = height1;
    this.age = age1;
}

var mapleTree = new Tree("Big Leaf Maple", 80, 50);
```

Creating Objects

- **Creating an object with prototypes (syntax):**

```
var obj = Object.create(Object.prototype [,properties]);
```

- **Examples:**

```
var myChild = Object.create(Object.prototype); //Object {}
myChild = Object.create({a:10, b:30}); // Object {a=10, b=30}

var myObj = {
    a : 10,
    b : 30
};
myChild = Object.create(myObj);
myChild.a;        //10
myChild.b;        //30
```

Accessing Object Properties

```
var myObject = {
    name: "luggage",
    length: 75,
    specs: {
      material: "leather",
      waterProof: true
    }
}
```

- **Getting and setting properties:**

```
myObject["name"];                    // "luggage"
myObject.name;                       // "luggage"
myObject.specs.material;             // "leather"
myObject["specs"]["material"];       // "leather"
myObject.width;                      // undefined
myObject.tags.number;                // TypeError thrown
```

```
myObject["name"] = "suitcase";       // name : "suitcase"
myObject.name = "suitcase";          // name : "suitcase"
myObject.width = 40;                 // creates a new property
myObject.tags.number = 6;            // TypeError thrown
```

Working with Object Properties

- Deleting properties:

```
delete myObject.length;
delete myObject.height;
```

true, even though it doesn't exist in the object

- Testing and enumerating properties:

```
var myObject = {
    one: "property value one",
    two: "property value two"
}
```

```
for (var myVar in myObject) {
    if (myObject.hasOwnProperty(myVar)) {
      console.log(myVar);
    }
}
```

Accessing Object Properties

- **Property `get` and `set`:**

```
var obj = {
    a : 45,
   get double_a() {return this.a * 2},
   set modify_a(x) {this.a -= x;}
};
```

```
obj.a;            // 45
obj.double_a;     // 90
obj.modify_a = 40;
obj.a;            // 5
obj.double_a;     // 10
```

Object Methods

- **Object method example:**

```
var myObj = {
    print: function() {
      console.log("Hello World!");
    }
};

myObj.print();
```

Topics

Part I:

- The JavaScript Grammar
- Variables and Types
- Expressions and Operators
- Statements and Loops

Part II:

- Objects
- Arrays
- Functions

Creating Arrays

- Initializing an array:

```
var myNewEmptyArray = [];

var numbers = [1, 2, 3, 4, 'five'];
```

- Creating an array with `new`:

```
var myNewEmptyArray = new Array();

var myNewNonEmptyArray = new Array(15);

var numbers = new Array(1, 2, 3, 4, 'five');
```

Accessing Array Elements

- **Length of an array:**

```
var c = ["blue","red","green","purple"];
c.length;  ← 4
c[20] = "white";
c.length;  ← 21
```

- **Iterating arrays:**

```
for (var i = 0; i < c.length; i += 1)
{
    console.log(c[i]);
}
```

```
for (var i in c)
{
    console.log(i);
}
```

Multidimensional Arrays

- Creating multidimensional arrays:

```
var myArr = new Array(10);
for (i = 0; i < 10; i++) {
    myArr[i] = new Array(10);
    for (j = 0; j < 10; j++) {
      myArr[i][j] = "[" + i + j + "]";
   }
}
```

```
var tic_tac_toe = [
[ 0, 0,'X'],
['X',0,'X'],
['X',0, 0]
];
```

Topics

Part I:

- The JavaScript Grammar
- Variables and Types
- Expressions and Operators
- Statements and Loops

Part II:

- Objects
- Arrays
- Functions

Defining Functions

- **Function definition:**

Name of the function

Arguments

```
function multiply(x, y) {
    var result = x * y;
    return result;
}
```

body

- **Calling a function:**

```
var total = multiply(135,6);
```

Anonymous Functions

```
var multiply = function (x, y) {
    var result = x * y;
    return result;
};

multiply(135, 6);
```

Calling a Function

- **Function as a method:**

```
var myObj = {
   name: "Hello World!",
    print: function(message) {
      console.log(this.name + message);
    }
};   // i.e. myObj.print(' JS Robot');
```

- **Function as a constructor:**

```
function Tree(type, height) {
    this.type = type;
    this.height = height;
}
Tree.prototype.getType = function() {
    return this.type;
}
var myTree = new Tree("Big Leaf Maple", 80);
myTree.getType();          // "Big Leaf Maple"
```

Calling a Function

- **Self-invoking:**

```
(function() { // Executed immediately })();
```

- **The `arguments` variable**

```
function sum() {
    var total = 0;
    for (var i= 0, l = arguments.length; i < l; i++) {
      total += arguments[i];
    }
    return total;
}
```

Recursion

- **Recursion to get the factorial of a number:**

```
function factorial(number) {
    if (number < 2)
      return 1;
    return number * factorial(number - 1);
}
```

- **Recursion to get the Fibonacci of a number:**

```
function fibonacci(number) {
    return (number < 2) ? number: fibonacci(number - 1) +
      fibonacci(number - 2);
}
```

Scopes

Try `window.x` in JSConsole.

- **Global Scope:**

```
var x = "I am a global var!";
function printVar()
{
    console.log(x);
}
```

- **Function scope:**

```
function declareVariable() {
    if (true) {
      var x = "Declared in a block of code";
    }
    console.log("Value of x: " + x);
}
```

Where Can I Learn More?

Resource	Website
ECMAScript Language Specification 5.1 Edition	http://www.ecma-international.org/ecma-262/5.1/
Douglas Crockford's JavaScript	http://javascript.crockford.com/
Mozilla Developer Network – JavaScript reference	https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

Summary

In this lesson, you should have learned how to:

- Write JavaScript code to:
  - Declare variables
  - Create an object
  - Create and call a function
  - Create and iterate arrays
- Write JavaScript arrays to store data
- Define JavaScript objects as a key-value store
- Access the properties of an object by using get and set