

JavaScript Basics

“Code with Passion!”



Topics

- What is and Why JavaScript?
- How and Where do you place JavaScript code?
- JavaScript language
 - > Variables, statements, code blocks, control flow
- JavaScript functions
 - > Defining functions – 3 different ways
 - > Calling functions
 - > Function as a method
- JavaScript data types
- JavaScript objects
 - > JavaScript object is a Hash
 - > 3 different ways of creating a JavaScript object

What is and Why JavaScript?

What is JavaScript?

- Scripting language
 - > Scripting language is a lightweight programming language
- Used to **add interactivity to HTML pages**
 - > JavaScript code could be embedded directly into HTML pages or in a separate file, which is referenced from the HTML page
- JavaScript is traditionally used as client-only language (within HTML page) – now slowly gaining some traction as a standalone or server-side language as well
 - > Example: Node.js
 - > Our focus in this codercamp is on the client side only

What can JavaScript do?

- JavaScript gives HTML page writers a programming tool for adding dynamic behavior
 - JavaScript can put dynamic text into an HTML page
 - JavaScript can react to events
 - JavaScript can read and write HTML elements
 - JavaScript can be used to validate input data
 - JavaScript can be used to detect the browser type & version
 - JavaScript can be used to detect whether a browser support a feature or not
 - JavaScript can be used to animate HTML elements
 - ...

Lab:

**Exercise 0: Install Chrome Browser &
Brackets (or VSC)
4262_javascript_basics.zip**





How and Where Do You Place JavaScript Code (in an HTML page)?

How to put JavaScript code into a HTML page?

- Use the `<script>` tag along with `type` attribute
- Scripts can be in the either `<head>` section and/or `<body>` section

```
<html>
<head>
<script type="text/javascript">
...
</script>
</head>
<body>
<script type="text/javascript">
...
</script>
</body>
</html>
```

Referencing External JavaScript File

- JavaScript code can be in a separate script file
- Script file can be provided locally or remotely
- Accessible via *src* attribute

```
<html>
<head>
<script language="JavaScript"
         type="text/javascript"
         src="http://somesite/myOwnJavaScript.js">
</script>
<script language="JavaScript"
         type="text/javascript"
         src="myOwnSubdirectory/myOwn2ndJavaScript.js">
</script>
```

remotely located

locally located

Lab:

**Exercise 1: JavaScript Code
4262_javascript_basics.zip**



JavaScript Language: **Variables**

JavaScript Variables

- You create a variable with or without the `var` keyword (scope will be different, however – explained in the following slide)

`var strname = <some value>;`

`strname = <some value>;`

- Variable names are case sensitive
 - > `yes` and `Yes` are two different variables
- Variable names must begin with a letter, the `$` character, or the underscore character
 - `myname`, `my_name`, `$myname`
- If you declare a variable without assigning any value to it, its type is *undefined*

`var myvar; // undefined type`

JavaScript Variable Scope

- In JavaScript, the variable scope is aligned with a function
 - > Not with a code block as in the case of C or Java
- Global scope variables (or global variables)
 - > If you declare a variable **outside a function**, it is in global scope
 - > All functions on the same page can access any global variables
 - > The usage of global variables are discouraged because it is prone to be overridden accidentally
- Local scope variables (or local variables)
 - > When you declare a variable with “var” and **within a function**, the variable can only be accessed within that function – the variable is in the local scope

Usage of “var” keyword & Variable Scope

- In the global scope, there's no difference between “var x” and “x” – they are both in global scope
- In the local scope – meaning inside a function, then "var" will create a local variable

```
// These are both global variables
var foo = 1;
bar = 2;

function() {
    var foo = 1; // Local scope variable
    bar = 2;    // Global scope variable

    // Execute an anonymous function
    (function() {
        var wibble = 1; // Local scope variable
        foo = 2; // Inherits from scope above (creating a closure)
        moo = 3; // Global scope variable
    }())
}
```

Lab:

Exercise 2: Variables
4262_javascript_basics.zip



JavaScript Language: **Statements & Code Blocks**

JavaScript Statements

- JavaScript statements are "instructions" to be "executed"
 - > JavaScript statements are often called JavaScript code
- Semicolons separate JavaScript statements (it is optional, however)

```
var x = 20;  
var y = 30;  
document.write("<h1>This is a heading</h1>");  
document.write("<p>This is a paragraph.</p>");  
document.write("<p>This is another paragraph.</p>");  
  
// The alert message below gets executed when the page is loaded  
alert("Hello Boston! This message gets displayed when a page is loaded.");
```

```
function displaymessage() {  
    alert("Hello World!");  
}
```

JavaScript Code Blocks

- JavaScript statements can be grouped together in a code block, inside curly brackets {...}
- The most common form of code block is a function

```
function displaymessage() {  
    var x = 20;  
    var y = 30;  
    alert("Result = " + (x+y));  
}
```

JavaScript Language: **Control flow**

JavaScript Language

- Conditional statement
 - > if, if.. else, switch
- Loop
 - > for loop, while loop, do-while loop
- try...catch
- throw

Lab:

Exercise 3: Control flow
4262_javascript_basics.zip



JavaScript Functions: Defining Functions

What is a Function?

- A function is a JavaScript procedure—a set of statements that performs a task or calculates a value
- A function can take 0 or more named parameters
- The function body can contain as many statements as you like, and can declare its own variables which are local to that function
 - > Variables with “var” keyword within the function are local scope variables
 - > Variables without “var” keyword within the function are global scope variables
- The return statement can be used to return a value at any time, terminating the function

Function Definitions (Declarations)

- A function can be defined (also called “declared”) in several ways
 1. Through function statement
 2. Through function expression
 3. Through function constructor (rarely used)
- When a function is declared, internally a function object is created, and that function object is assigned to a property of the owning object
 - > The owning object of the top-level function is “window” for browser
- Note that function definition (declaration) is just that – it is NOT function invocation (function execution)
 - > In other words, a function object gets created but it is not executed

#1: Through function Statement

- A function statement is made of *function* keyword, followed by
 - The optional name of the function
 - A list of arguments to the function
 - The JavaScript statements enclosed in curly braces, { ... }
- A function statement is a genuine JavaScript statement
 - Execution of the function statement creates a function object
- A function object, once created, is assigned to a property of owner object – the property name is the same as function name

```
// Declare a named function as a function statement.  
// “myNamedFunction” property of owner object points to  
// newly created function object.  
function myNamedFunction(arg1, arg2) {  
    console.log(arg1, arg2);  
}
```

#2: Through function Expression

- A function can be defined as a function expression
 - The function has to be assigned to a variable or can be passed as an argument in this case
- A function expression can be anonymous (name is optional)

```
// Create a function object via anonymous function expression and  
// assign it to myFunction1 variable  
var myFunction1 = function(something){  
    console.log(something);  
}
```

- A function object is created and then assigned to the property of the owning object – the property name is the variable name, *myFunction1* in the example above

#3: Through function Constructor (Rarely used)

- The *Function()* constructor expects any number of string arguments
- The last argument is the body of the function - it can contain arbitrary JavaScript statements, separated from each other by semicolons

```
// Create a function through Function Constructor
var my_func = new Function("x", "y", "return x+y;");
```

```
/* This is the same as above
function my_func(x, y){
    return x+y;
}
```

```
my_func(10,20);
```

JavaScript Functions: A Function as a Method

A Function as a Method

- A property of a JavaScript object whose value is a function object is called a method

```
// Declare a function
function functionDefinedSomewhere(something) {
    console.log(something)
}

// Create a JavaScript object
var myPerson = {
    firstname : "John",
    lastname : "Doe",
    age : 50,
    tellYourage : function() { // Anonymous function without argument
        console.log("The age is " + this.age);
    },
    tellSomething : function(something) { // Anonymous function with an argument
        console.log(something);
    },
    tellWhatever : functionDefinedSomewhere // Named function
}
```

tellYourage, tellSomething, and tellWhatever methods

JavaScript Functions: **Function Invocation** **(Function execution)**

Function Invocation (Function Execution)

- Defining a function does not invoke(execute, call) it automatically
 - > Defining the function simply creates a function object and assigns it to a property of owning object
 - > In order to execute the function (perform some task), you have to explicitly invoke it
- A function gets executed only by an invocation (or by an event if the function is configured as event handler)
 - > In order to prevent the browser from executing a script as soon as the page is loaded, you want to write your script as a function
- You may invoke a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file)

Function Invocation (Function Execution)

```
<script type="text/javascript">
```

```
// Declare/define a named function as a function statement  
function myNamedFunction(something) {  
    console.log(something);  
}
```

Function definition

```
// Declare/define an anonymous function expression and assign it to
```

```
// myFunction1 variable  
var myFunction1 = function(something){  
    console.log(something);  
}
```

Function definition

```
var myFunction2 = myNamedFunction;
```

// Invoke/execute/call functions

```
myNamedFunction("Life is good!");  
myFunction1("Hello!");  
myFunction2("Goodbye!");
```

Function invocation

```
</script>
```

Function Invocation via Event

```
<html>
<head>
<title/>
<script type="text/javascript">
    // If alert("Hello world!!") below had not been written within a
    // function, it would have been executed as soon as the page gets loaded.
    function displaymessage() {
        alert("Hello World!")
    }
</script>
</head>

<body>
<form>
<input type="button" value="Click me!">
    onclick="displaymessage()"
</form>
</body>
</html>
```

Function invocation via event

Lab:

Exercise 4: Defining and Invoking functions
4262_javascript_basics.zip



JavaScript Data Types

JavaScript Data Types

- JavaScript is a loosely typed or dynamic type language
 - > You don't have to declare a type of a variable
 - > The type gets determined automatically when the program gets executed
 - > You can use a same variable as different types

```
var foo = 35;          // foo is Number type
var foo = "passion";  // foo is String type
var foo = true;        // foo is Boolean type
```

JavaScript Data Types

- There are 7 data types
- 6 data types are primitive types
 - > Boolean, Null, Undefined, Number, String, Symbol
 - > Primitive types define immutable values (values, which are incapable of being changed) - these immutable values of the primitive types are called as "primitive values"
- The remaining data type is Object type

Primitive types

- Boolean type
 - > Can have two values: *true* or *false*
- Null type
 - > Has exactly one value: *null*
- Undefined type
 - > A variable that has not been assigned a value has the value *undefined*
- Number type
- String type
- Symbol type (introduced in ECMAScript 6)
 - > Unique and immutable

Object Type

- A JavaScript object has properties and methods
 - Example: *String* JavaScript object has *length* property and *toUpperCase()* method

```
<script type="text/javascript">  
  
var txt="Hello World!"  
document.write(txt.length)  
document.write(txt.toUpperCase())  
  
</script>
```

Lab:

Exercise 5: Object types
4262_javascript_basics.zip



JavaScript Objects: **Hash (Associative Array)**

JavaScript Object is a Hash

- A JavaScript object is essentially a hash (an associative array) with property-name/value pairs
 - > Property name has to be unique
 - > It is like a Map in Java
 - > There is no exception to this rule - even a function object is a hash

```
{  
    name1: value1,  
    name2: value2,  
    name3: value3,  
    ....  
    nameN: valueN  
}
```

How to Refer to Property Names

- The following two lines of code are semantically equivalent

```
myObject.myfield = "something";  
myObject['myfield'] = "something";
```

- [..] notation can take variable, however

```
var x = "test";  
myObject[x] = "Passion!";  
console.log(myPerson.test); // Passion!
```

Value of a Property Can Be function object

```
var myPerson = {  
    firstname: "John",  
    lastname: "Doe",  
    age: 50,  
    tellYourage: function () { // Anonymous function without argument  
        alert("The age is " + this.age );  
    },  
    tellSomething: function(something) { // Anonymous function with an argument  
        alert(something);  
    },  
    tellWhatever: functionDefinedSomewhere // Named function  
}  
  
function functionDefinedSomewhere(something){  
    alert(something)  
}  
  
myPerson.tellYourage();  
myPerson.tellSomething("Life is good!");  
myPerson.tellWhatever("Hello");
```

The value of the tellSomething property is a function object

Value can be another Java Script Object

```
var myVar = {  
    count: 20,  
    person: myPerson // myPerson was defined in previous slide  
}  
  
myVar.person.tellSomething("Life is REALLY REALLY good!");
```

JavaScript Object vs. Java Object

- Similarities
 - > Both has properties and methods
- Differences
 - > JavaScript object can be dynamically typed (while in Java, object is statically typed)
 - > In JavaScript, properties and methods can be added dynamically to a JavaScript object during runtime (while in Java, properties and methods need to be defined at compile time)
 - > In JavaScript, a method is defined by assigning a function object to a property

JavaScript Objects; 3 Different Ways of Creating a JavaScript Object

3 Ways of Creating a JavaScript Object

1. Create an object instance as Hash Literal (You have already seen this) – preferred
2. Define a function as a Constructor first and then create an instance of an object from it
3. Create a direct instance of an object by using built-in constructor of the built-in “Object” object

Option #1: Creating JavaScript Object as a Hash Literal

```
// Create JavaScript object as a Hash Literal then assign to "personObj"
var personObj = {
    firstname: "John",
    lastname: "Doe",
    age: 50,
    tellYourage: function () {
        alert("The age is " + this.age );
    }
    tellSomething: function(something) {
        alert(something);
    }
}

// Call methods of "personObj" JavaScript object
personObj.tellYourage();
personObj.tellSomething("Life is good!");
```

Option #2: Create from a Constructor Function (Template)

- A function defines the structure of a JavaScript object – it plays a role of a template

```
// Define a Constructor function
function Person(firstname,lastname,age,eyecolor){
    this.firstname=firstname;
    this.lastname=lastname;
    this.age=age;
    this.tellYourage=function(){
        alert("This age is " + this.age);
    }
}
```

// Continued in the next slide

Option #2: Create from a Constructor Function (Continued)

- Once you have a Constructor function (as you saw in the previous slide), you can create new instances of JavaScript object using *new* keyword

```
myFather=new Person("John","Doe",50,"blue");
myMother=new Person("Sally","Rally",48,"green");
```

- You can then add new properties and functions to new objects

```
myFather.newField = "some data";
myFather.myfunction = function() {
    alert(this["fullName"] + " is " + this.age);
}
```

Option #3: Create a Direct Instance of a JavaScript Object from “Object” object

- By invoking the built-in constructor for the *Object* object

```
// Initially empty with no properties or methods  
personObj=new Object(); // same as personObj = {}
```

- Add properties to it

```
personObj.firstname="John";  
personObj.age=50;
```

- Add an anonymous function to the *personObj*

```
personObj.tellYourage=function(){  
    alert("This age is " + this.age);  
}  
  
// You can call then tellYourage function as following  
personObj.tellYourage();
```

Option #3: Create a Direct Instance of a JavaScript Object from “Object” (Continued)

- Add a pre-defined function

```
function tellYourage(){  
    alert("The age is" + this.age);  
}  
  
personObj.tellYourage=tellYourage;
```

- By the way, note that the following two lines of code are doing completely different things

```
// Set property with a function  
personObj.tellYourage=tellYourage;  
  
// Set property with returned value of the function  
personObj.tellYourage=tellYourage();
```

Lab:

Exercise 6: Create objects
4262_javascript_basics.zip



Code with Passion!

