What is Javascript

•Scripting language of browser •Dynamic •Object oriented and functional features •Java syntax •Small and simple

Why Javascript

•Easy to learn and edit •Prototyping language •Easy to debug•Object oriented design

History

•Developed by Brendan Eich •Netscape(2.0) 1995 •Microsoft Jscript

•ECMA (**European Computer Manufacturer's Association**)script.

Writing javascripts

•DOM helps Javascript to interact with html. •Javascript displays data in different ways •Using **console.log()**. Variables scope •Global Scope •Block vs Function Scope •Variable hoisting

Strict mode •Introduced in ECMA5 •Strict mode applies to *entire scripts* or to *individual functions.* •Not applied to blocks •Applied by “use strict” •Catches coding bloopers and throws exceptions •Prevents from unsafe actions.

objects

•Objects are collection of properties

•Objects can have single value or multiple values

•var car=“volvo”;

•var car= {type:" Volvo", model:“100", color: "white"};

•Objects with name value pair are called properties.

Functions

•First class citizen

•Can assign to a variable

•Can be added as an attribute to an object

•Can be passed as a parameter to a function

•Can be returned from a function.

•Can be passed as an expression

Closures 13

•JavaScript allows writing nested functions i.e. function within a function

•The inner functions can access variables defined outside the inner function and this concept is called a Closure.

•Closures are functions that refer to independent (free) variables.

•The function defined in the closure 'remembers' the environment in which it was created.

•A closure is a function defined within another scope that has access to all the variables within the outer scope.

Inheritance

•Each object has an internal link to another object called its **prototype**.

•JavaScript follows prototypal inheritance.

•One object can inherit properties of other object

•Every object can then be used as a prototype for another object.

IIFE(Immediately invoked function) 15

•IIFE is an design pattern

•Anonymous function created and then immediately invoked

•Can be used to avoid variable hoisting from within blocks

•Primary reason to use an IIFE is to obtain data privacy

•Any variables declared within the IIFE cannot be accessed by the outside world.

•Syntax:-

(function() {

// the code here is executed once in its own scope

})();

**Variable Hoisting**  
All variable declarations are hoisted (lifted and declared) to the top of the function, if defined in a function, or the top of the global context, if outside a function.

It is important to know that only variable declarations are hoisted to the top, not variable initialization or assignments (when the variable is assigned a value).

In strict mode, an error will occur if you assign a variable a value without first declaring the variable. Always declare your variables.

**Variable Scope**  
A variable’s scope is the context in which the variable exists. The scope specifies from where you can access a variable and whether you have access to the variable in that context.

**Variables have either a local scope or a global scope.**

**Local Variables (Function-level scope)**  
Unlike most programming languages, JavaScript does not have block-level scope (variables scoped to surrounding curly brackets); instead, JavaScript has function-level scope. Variables declared within a function are local variables and are only accessible within that function or by functions inside that function. See my post on [Closures](http://javascriptissexy.com/javascript-closures-in-lovely-detail/) for more on accessing variables in outer functions from inner functions.

Demonstration of Function-Level Scope

|  |  |
| --- | --- |
|  | var name = "Richard"; |
|  | ​ |
|  | ​function showName () { |
|  | var name = "Jack"; // local variable; only accessible in this showName function​ |
|  | console.log (name); // Jack​ |
|  | } |
|  | console.log (name); // Richard: the global variable  Local Variables Have Priority Over Global Variables in Functions If you declare a global variable and a local variable with the same name, the local variable will have priority when you attempt to use the variable inside a function (local scope):   |  |  | | --- | --- | |  | var name = "Paul"; | |  | ​ | |  | ​function users () { | |  | // Here, the name variable is local and it takes precedence over the same name variable in the global scope​ | |  | ​var name = "Jack"; | |  | ​ | |  | ​// The search for name starts right here inside the function before it attempts to look outside the function in the global scope​ | |  | console.log (name); | |  | } | |  | ​ | |  | users (); // Jack | |

**Global Variables**

* Any variable declared or initialized outside a function is a global variable, and it is therefore available to the entire application. For example:

|  |  |
| --- | --- |
|  | // To declare a global variable, you could do any of the following:​ |
|  | ​var myName = "Richard"; |
|  | ​ |
|  | ​// or even​ |
|  | firstName = "Richard"; |
|  | ​ |
|  | ​// or ​ |
|  | ​var name; //​ |
|  | name; |
|  | ​</pre> |

* If a variable is initialized (assigned a value) without first being declared with the var keyword, it is automatically added to the global context and it is thus a global variable:

|  |  |
| --- | --- |
|  | function showAge () { |
|  | // Age is a global variable because it was not declared with the var keyword inside this function​ |
|  | age = 90; |
|  | console.log(age);// ​ |
|  | } |
|  | ​ |
|  | showAge (); // 90​ |
|  | ​ |
|  | ​// Age is in the global context, so it is available here, too​ |
|  | console.log(age); // 90 |