JavaScript Advance Notes

# Objects

* There are different ways to create new objects:

1. Using an **object literal**.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

1. using **keyword** **new**.

var person = **new Object ();**

person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

1. Define an **object constructor**, and then create objects of the constructed type.

function person(first, last, age, eye) {  
    **this**.firstName = first;  
    **this**.lastName = last;  
    **this**.age = age;  
    **this**.eyeColor = eye;  
}// This is known as construction function  
var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");

|  |  |
| --- | --- |
| **Note** | In ECMAScript 5, an object can also be created with the **function Object.create**(). |

## Objects are Mutable

* They are addressed by reference, not by value.
* The object x is not a **copy** of y. It **is** y. Both x and y points to the same object.

var x = y;  // This will not create a copy of y.

## Prototype Properties

* JavaScript objects **inherit the properties & methods** of their prototype.
* The prototype is also **an object**.
* The delete keyword does not delete inherited properties, but if you **delete a prototype property**, it will **affect all objects inherited** from the prototype.
* Above 3rd example is creating a prototype, that is, the **constructor function is the** prototype for your person objects.

## Adding a Property/Method to an Object

* The property/method will be added to myFather. Not to myMother. Not to any other person objects.

myFather.nationality = "English"; //The prop will be added to myFather. Not to myMother.

myFather.name = function () {//The method will be added to myFather. Not to myMother.  
     return this.firstName + " " + this.lastName;

};

## Adding Properties to a Prototype or function constructor

* To add a new property to a constructor, you must add it to the constructor function:

function person(first, last, age, eyecolor) {  
    **this**.firstName = first;  
    **this**.lastName = last;  
    **this**.age = age;  
    **this**.eyeColor = eyecolor;  
    **this**.nationality = "English";//Adding property to prototype, now available to all objects

**this**.name =

function() {return this.firstName + " " + this.lastName;};//Adding Method to prototype  
 }

## Or can be added through prototype Property

person.**prototype**.nationality = "English";

person.**prototype**.name = function() {  
 return this.firstName + " " + this.lastName;

}

lamp copy.pngOnly modify your **own** prototypes. Never modify the prototypes of standard JavaScript objects.

# Function Definitions

## Declarations

## Normal Declarations

function functionName(parameters) { code to be executed }

## Function Expressions

var x = function (a, b) {return a \* b}; // semi-colon is required

## Function () Constructor

var myFunction = new Function("a", "b", "return a \* b");

## Self-Invoking Functions

(function () {  
    var x = "Hello!!";      // I will invoke myself, known as  anonymous self-invoking function  
})();

## Functions are Objects

* The **typeof** operator returns "function" for functions.
* Functions are **objects** and have both **properties** and **methods**.
* The **arguments.length** property returns the number of arguments received when the function was invoked:

function myFunction(a, b) {  
    return **arguments.length**; // 2 -This is the length of arguments passed in calling function.  
}

* Function **parameters** are the **names** listed in the function definition.
* Function **arguments** are the real **values** passed to (and received by) the function.

## Arguments Object

* built-in object
* argument object contains an array of the arguments used when the function was called (invoked)
* **arguments.length** returns the length of arguments of function invoked, not the parameters passed in function declaration.
* **Arguments are Passed by Value**
* **Objects are Passed by Reference**

## Parameter Rules

* Function definitions **do not specify data types** for parameters.
* Functions **do not perform type checking** on the passed arguments.
* Functions **do not check the number of arguments** received.

## 4 Ways to Invoke a Functions

1. **As a Function**

function myFunction(a, b) {  
    return a \* b;  
}  
myFunction(10, 2);           // myFunction(10, 2) will return 20

window.myFunction(10, 2);    // window is global object

1. **Global Object**

In a web browser the global object is the browser window.

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this.firstName + " " + this.lastName;  
    }  
}  
myObject.fullName();         // Will return "John Doe"

1. **With a Function Constructor**

// This is a function constructor:  
function myFunction(arg1, arg2) {  
    this.firstName = arg1;  
    this.lastName  = arg2;  
}  
// This creates a new object  
var x = new myFunction("John","Doe");  
x.firstName;   // Will return "John"

1. **With a Function Method**

* **call()** and **apply()** are predefined JavaScript function methods. Both methods can be used to invoke a function.

function myFunction(a, b) {  
    return a \* b;  
}  
myObject = myFunction.**call**(myObject, 10, 2);     // Will return 20

myArray = [10, 2];  
myObject = myFunction.**apply**(myObject, myArray);  // Will also return 20

* Both methods takes an **owner object as the first argument**. The only difference is that **call()** takes the function ***arguments separately***, and **apply()** takes the function ***arguments in an array***.

## Closures

* Remember **self-invoking functions**? **Singleton Class**

var add = (function () {  
    var counter = 0; // function allow to make variables **private**  
    return function () {return counter += 1;}//assigned to add return value of a self-invoking function  
})();  
  
add();  
add();  
add();  
  
// the counter is now 3

* A closure is a **function having access to the parent scope**, **even after the parent function has closed**.

# HTML DOM (Document Object Model)

* When a web page is loaded, the browser creates a **DOM** of the page.
* DOM is created as a tree of **objects**.
* It gives power to create dynamic HTML.



* "The W3C Document Object Model (DOM) is a **platform and language-neutral interface** that allows programs and scripts to **dynamically access and update** the content, structure, and style of a document."
* The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

## Finding HTML Elements

|  |  |
| --- | --- |
| Method | Description |
| document.getElementById() | Find an element by element id |
| document.getElementsByTagName() | Find elements by tag name |
| document.getElementsByClassName() | Find elements by class name |
| document.querySelectorAll() | matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc) eg. *"p.intro" of element p with class intro* |

## Changing HTML Elements

|  |  |
| --- | --- |
| Method | Description |
| element.innerHTML= | Change the inner HTML of an element |
| element.attribute= | Change the attribute of an HTML element |
| element.setAttribute(attribute,value) | Change the attribute of an HTML element |
| element.style.property= | Change the style of an HTML element |

## Adding and Deleting Elements

|  |  |
| --- | --- |
| Method | Description |
| document.createElement() | Create an HTML element |
| document.removeChild() | Remove an HTML element |
| document.appendChild() | Add an HTML element |
| document.replaceChild() | Replace an HTML element |
| document.write(text) | Write into the HTML output stream. It will overwrite the document, so don’t use after document load. |

## Adding Events Handlers

|  |  |
| --- | --- |
| Method | Description |
| document.getElementById(id).onclick=function(){code} | Adding event handler code to an onclick event |

## Document Objects Properties

<http://www.w3schools.com/js/js_htmldom_document.asp>

anchors

applets (Not in HTML5)

baseURI

body

cookie

doctype

documentElement <html>

documentMode

documentURI

domain

embeds

forms

head

images

implementation

inputEncoding

lastModified

links

readyState

referrer

scripts

strictErrorChecking

title

URL

## Style Object Properties

<http://www.w3schools.com/jsref/dom_obj_style.asp>

The onload and onunload Events

* The onload and onunload events are triggered when the user enters or leaves the page.
* The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
* The onload and onunload events can be used to deal with cookies.

The onmousedown, onmouseup and onclick Events

* The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

<http://www.w3schools.com/jsref/dom_obj_event.asp>

addEventListener

* The addEventListener() method attaches an event handler to the specified element.
* The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
* You can add many event handlers to one element.
* You can add many event handlers of the same type to one element, i.e two "click" events.
* You can add event listeners to any DOM object not only HTML elements. i.e the window object.
* The addEventListener() method makes it easier to control how the event reacts to bubbling.
* When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
* You can easily remove an event listener by using the removeEventListener() method.

element.addEventListener(event, function, useCapture);

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM, bubbling and capturing.

Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?

In bubblingthe inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.

In capturingthe outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

addEventListener(event, function, useCapture);

The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.