Virtuoso Infotech Pvt. Ltd.

Functions

- A function is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing the same code again and again.
- It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.
- JavaScript function are of two types built in function and User defined function.

Built In function

Functions	Description
isNaN	This function is intended to determines whether value is a legal number or not.
parseInt/parseFloat	This function is intended to converts string value to integer/float.
eval	This function is intended to execute Javascript source code.
number	This function is intended to converts object to the corresponding number value.
string	This function is intended to converts object to the corresponding string value.
encodeURI/DecodeURI	This function is intended to encode/deccode URI.
decodeURI	This function is intended to decode URI.

Parse Int function

The parseInt() function in Javascript parses a string value and returns an integer value

```
var str = "90“
 var num = 10;
var sum = str + num; //before use parseInt()
alert ("Sum is : "+ sum);
sum = parseInt(str) + num; //using parseInt()
alert ("Sum is : "+ sum);
```

When you execute the code, first it will alert "Sum is : 9010" and after using parseInt(), it will alert "Sum is : 100

Eval function

- The eval() function evaluates or executes an argument.
- If the argument is an expression, eval() evaluates the expression. If the argument is one or more JavaScript statements, eval() executes the statements.

```
Eval(new String(“2+2”)); // returns a string object containing “2+2”
Eval(“2+2”);  // returns 4
function test(){
  var x = 2, y = 4;
  console.log(eval("x + y"));  // Direct call, uses local scope, result is 6
  var geval = eval;
  console.log(geval("x + y")); // Indirect call, throws ReferenceError because
  `x` is undefined}
```

isFinite function

- The **isFinite** is used to determine whether a specified number is finite or not. **isFinite** is a top-level function and is not associated with any object.

- **Note** that the function returns false if the argument is NAN, positive infinity or negative infinity otherwise it returns true.

- The following example shows how to use isFinite() function.

```
console.log(isFinite("Good Morning"));  // False
console.log(isFinite("2009/01/01"));   // False
console.log(isFinite(-9.34));  //True
console.log(isFinite(15-12));  //True
console.log(isFinite(0));  //True
```

User Defined function

```
<script type="text/javascript">
function <function name> (argument list.....)
{
Statements .....
}
</script>
```

Function with/without Parameters

```
<INPUT TYPE="button" VALUE="Show Message" onClick="showMessage();">

function showMessage()
{
   alert('Hello Javascript');
}

<INPUT TYPE="button" VALUE="Show Message" onClick=" add(10, 20);">

function add(a, b)
{
var c;
c= a + b;
Return c;
}
```

Hoisting

- A variable can be used before it has been declared.

- x = 5; // Assign 5 to x

```
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                     // Display x in the element

var x; // Declare x
```

- Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

Hoisting

- JavaScript Initializations are Not Hoisted
- Example 1 :: var x = 5; // Initialize x

```
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x + " " + y;           // Display x and y
var y = 7; // Initialize y
```

- Because of hoisting, y has been declared before it is used, but because initializations are not hoisted, the value of y is undefined.
- Example 2 :: var x = 5; // Initialize x

```
var y;     // Declare y
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x + " " + y;           // Display x and y
y = 7;    // Assign 7 to y
```

Function Hoisting

- **Function Hoisting :** By default, JavaScript moves all the function declarations to the top of the current scope. This is called function hoisting.
- This is the reason JavaScript functions can be called before they are declared.

- var sum = addNumbers(10, 20);
  document.write(sum);

  function addNumbers(firstNumber, secondNumber)
  {
  var result = firstNumber + secondNumber;
  return result;
  }

Anonymous Function

- An anonymous function is a function that was declared without any named identifier to refer to it.
- var add = function (firstNumber, secondNumber)

```
{
    var result = firstNumber + secondNumber;
    return result;
}

var sum = add(10, 20);
document.write(sum);
```

Closure

- A closure is an inner function that has access to its own variables, it has access to the outer function’s variables, and it has access to the global variables.

```
function sayHello2(name) {
    var text = 'Hello ' + name; // Local variable
    var say = function() { console.log(text); }
    return say;
}
var say2 = sayHello2('Bob');
say2(); // logs "Hello Bob"
```

Self Invoking Function

```
var result = (function computeFactorial(number)
{
    if (number <= 1)
    {
        return 1;
    }
    return number * computeFactorial(number - 1);
})(5);

document.write(result);
```

Output : 120

Cookies

- Web Browsers and Servers use HTTP protocol to communicate and HTTP is a stateless protocol.

- But for a commercial website, it is required to maintain session information among different pages.

- A JavaScript cookie is nothing but a small piece of data that's stored in user's browser and can be picked up by the web pages when it requires

Cookies

Cookies are a plain text data record of 5 variable-length fields ::

- **Expires** − The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain** − The domain name of your site.
- **Path** − The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure** − If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value** − Cookies are set and retrieved in the form of key-value pairs

Create Cookie

Just assign the string you want for the cookie to the document.cookie property.

Example ::

```
document.cookie = "key1=value1;key2=value2;expires=date";
```

Create Cookie ::

```
function createCookie(name,value,days) {
    if (days) {
        var date = new Date();
        date.setTime(date.getTime()+(days*24*60*60*1000));
        var expires = "; expires="+date.toGMTString();
    }
    else var expires = "";
    document.cookie = name+"="+value+expires+"; path=/";
}

//call above function to set the cookies
createCookie('VITcookie','testcookie',7);
```

Read Cookie

To read a cookie, you need cookies name to read back from browser.

```
function readCookie ( cookie_name )
{
  var cookie_string = document.cookie ;
  if (cookie_string.length != 0) {
    var cookie_value =
    cookie_string.match ( '(^|;)[\s]*' + cookie_name + '=([^;]*)' );
    return decodeURIComponent ( cookie_value[2] ) ;
  }
  return '';
}

//call above function to read a cookie
readCookie('VITcookie');
```

Delete Cookie

```
function deleteCookie(name)
{
    createCookie(name,"",-1);
}

//call above function to read a cookie
deleteCookie('pkcookie');
```

Page Redirect

URL redirection, also called URL forwarding, is a way to automatically redirect a web page to another web page. The redirected page is often on the same website, or it can be on a different web site or a web server. - See more at: http://www.corelangs.com/js/basics/redirect.html#sthash.dkyxmxI1.dpuf

- There are 3 ways to redirect user from one page to another.

```
function Redirect() {
    window.location="http://www.tutorialspoint.com";
    window.navigate("index.php");
    self.location="index.php";
     top.location="error.php";  }
```

Window.Location

- ***window.location.href*** returns the href (URL) of the current page

http://www.w3schools.com/js/js_window_location.asp

- ***window.location.hostname*** returns the domain name of the web host

www.w3schools.com

- ***window.location.pathname*** returns the path and filename of the current page

/js/js_window_location.asp

Window.Location

- ***window.location.protocol*** returns the web protocol used (http:// or https://)

```
http:
```

- ***window.location.assign*** method loads a new document

```
<input type="button" value="Load new document" onclick="newDoc()">

function newDoc() {
    window.location.assign("http://www.w3schools.com")
}
```

Window.Location

window. location.replace() Method replaces the current document with a new one. In replace() method, you can pass new URL to replace() method and it will perform an HTTP redirect

window.location.replace("http://www.virtuosoitech.com”)

The window.navigate() method is similar to assigning a new value to the window.location.href property. Because it is only available in MS Internet Explorer, so you should avoid using this in cross-browser development.

window.navigate("http:// www.virtuosoitech.com ");

Page Refresh

- You can refresh a web page using JavaScript **location.reload** method.
- This code can be called automatically upon an event or simply when the user clicks on a link.
- Refresh page
- You can also use JavaScript to refresh the page automatically after a given time period. Here **setTimeout()** is a built-in JavaScript function which can be used to execute another function after a given time interval.

```
Function AutoRefresh(t) {
setTimeout("location.reload(true);",t); }

onload="javascript:AutoRefresh(5000);"
```

Dialog Boxes

- An alert dialog box is mostly used to give a warning message to the users.

```
function Warn() {
        alert ("This is a warning message!");
        document.write ("This is a warning message!");
    }
```

- A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: Ok and Cancel.

```
function getConfirmation(){
        var retVal = confirm("Do you want to continue ?");
        if( retVal == true ){
          document.write ("User wants to continue!");
          return true;
        }
        else{
          document.write ("User does not want to continue!");
          return false;
        }
    }
```

- The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

Array

- The **Array** object lets you store multiple values in a single variable.

- It stores a fixed-size sequential collection of elements of the same type.

- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

- There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

Array Literal

- The syntax of creating array using array literal is given below:

```
var arrayname=[value1,value2.....valueN];
```

```
<script>
var emp=[“Sunil","Vimal","Ratan"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}
</script>
```

OUTPUT ::

Sunil
Vimal
Ratan

Array New Keyboard

- The syntax of creating array using array new keyword is given below:

```
var arrayname=new Array();
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

Array Constructor

- Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

```
<script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

Array Properties

Functions	Description
forEach()	Calls a function for each element in the array.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
concat()	Joins two or more arrays, and returns a copy of the joined arrays
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reduce()	Reduce the values of an array to a single value (going left-to-right)
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.

Array Properties

Functions	Description
slice()	Selects a part of an array, and returns the new array
some()	Checks if any of the elements in an array pass a test
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
valueOf()	Returns the primitive value of an array

IndexOf

- The index of the first occurrence of *searchElement* in the array, or -1 if *searchElement* is not found.

- Syntax :: array1.indexOf(searchElement[, fromIndex])

- The array elements are compared to the *searchElement* value by strict equality, similar to the === operator

- The optional *fromIndex* argument specifies the array index at which to begin the search. If *fromIndex* is greater than or equal to the array length, -1 is returned. If *fromIndex* is negative, the search starts at the array length plus *fromIndex*.

IndexOf

```
// Create an array. (The elements start at index 0.)
var ar = ["ab", "cd", "ef", “pq", "cd"];

// Determine the first location of "cd".
document.write(ar.indexOf("cd") + "<br/>");
// Output: 1

// Find "cd" starting at index 2.
document.write(ar.indexOf("cd", 2) + "<br/>");
// Output: 4

// Find "gh" (which is not found).
document.write (ar.indexOf("gh")+ "<br/>");
// Output: -1
```

ValueOf

- Returns the primitive value of the specified object.
- Syntax :: array.valueOf()

```
var arr = [1, 2, 3, 4];
var s = arr.valueOf();

if (arr === s)
document.write("same");
else
document.write("different");

// Output:
// same
```

Pop/Push Method

- **POP() Method**
- Removes the last element from an array and returns it.

  Example :: var fruits = [“Grapes", "Orange", "Apple", "Mango"];

  fruits.pop();

- Output :: Grapes,Orange,Apple

- **PUSH() Method**
- Add a new item to an array and returns it.

  Example :: var fruits = [" Grapes ", "Orange", "Apple", "Mango"];

  fruits.push("Kiwi");

  Output :: Grapes,Orange,Apple,Mango,Kiwi.

Reverse/Shift Method

- **Reverse() Method**
- Reverse the order of the elements in an array.

  Example :: var fruits = [“Grapes", "Orange", "Apple", "Mango"];
  fruits.pop();

  Output :: Mango,Apple,Orange,Grapes

- **Shift() Method**
- Remove the first item of an array:

  Example :: var fruits = [“Grapes", "Orange", "Apple", "Mango"];
  fruits.shift();

  Output :: Orange,Apple,Mango

Splice/Slice Method

- **Slice() Method**
- Select elements from an array.
- The slice() method selects the elements starting at the given *start* argument, and ends at, *but does not include*, the given *end* argument.

  Example :: var fruits = [“Grapes", "Orange", "Lemon", "Apple", "Mango"];
  var citrus = fruits.slice(1, 3);

  Output :: Orange,Lemon
- **Splice() Method**
- Add items to the array.
- The splice() method adds/removes items to/from an array, and returns the removed item(s). 2: At what index to add, 0 : How many elements to remove.

  Example :: var fruits = [“Grapes", "Orange", "Apple", "Mango"];
  fruits.splice(2, 0, "Lemon", "Kiwi");

  Output :: Grapes,Orange,Lemon,Kiwi,Apple,Mango

Date

- The Date object is used to work with dates and times.
- Date objects are created with new Date().
- There are four ways of instantiating a date:

new Date()
new Date(dateString)
new Date(milliseconds)
new Date(year, month, day, hours, minutes, seconds, milliseconds)

- **new Date()**

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

Output :: Fri Oct 14 2016 06:35:32 GMT+0530 (India Standard Time)

Date

- **new Date(milliseconds)**

```
var d = new Date(86400000);
document.getElementById("demo").innerHTML = d;
```

Output :: Fri Jan 02 1970 05:30:00 GMT+0530 (India Standard Time)

- **new Date(dateString)**

```
var d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
```

Output :: Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)

- **new Date(year, month, day, hours, minutes, seconds, milliseconds)**

```
var d = new Date(99,5,24,11,33,30,0);
document.getElementById("demo").innerHTML = d;
```

Output :: Thu Jun 24 1999 11:33:30 GMT+0530 (India Standard Time)

Date Functions

Functions	Description
Date.now	Returns the number of milliseconds between January 1, 1970, and the current date and time
Date.parse	Parses a string containing a date, and returns the number of milliseconds between that date and midnight, January 1, 1970.
Date.UTC	Returns the number of milliseconds between midnight, January 1, 1970 Universal Coordinated Time (UTC) (or GMT) and the supplied date.

Date Functions

- **Date.now()**

  var n = Date.now();

  Output :: 1476408265294

- **Date.parse()**

  var d = Date.parse("March 21, 2012");

  Output :: 1332268200000

- **Date.UTC()**

  var d = Date.UTC(2012,02,30);

  Output :: 1333065600000

Date Functions

Functions	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getMonth()	Get the month (0-11)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the seconds (0-59)
getSeconds()	Get the seconds (0-59)

Document Object Model(DOM)

- The **document object** represents the whole html document.
- When html document is loaded in the browser, it becomes a document object.
- It is the **root element** that represents the html document.
- It has properties and methods. By the help of document object, we can add dynamic content to our web page.

getElementById() Method

- The getElementById() method returns the element that has the ID attribute with the specified value.
- Returns *null* if no elements with the specified ID exists.
- An ID should be unique within a page. However, if more than one element with the specified ID exists, the getElementById() method returns the first element in the source code.

getElementById() Method

```
<p id="demo">Click the button to change the color of this paragraph.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
    var x = document.getElementById("demo");
    x.style.color = "red";
}
```

getElementByTagName() Method

- The getElementsByTagName() method returns a collection of all elements in the document with the specified tag name, as a NodeList object.
- The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

- **Tip:** The parametervalue "*" returns all elements in the document.

- **Tip:** You can use the length property of the NodeList object to determine the number of elements with the specified tag name, then you can loop through all elements and extract the info you want.

getElementByTagName() Method

```
<div id="myDIV">
  <p>First p element in div (index 0).</p>
  <p>Second p element in div (index 1).</p>
  <p>Third p element in div (index 2).</p>
</div>

<p>Click the button to add a background color to the second p element inside the div element.</p>
<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
    var x = document.getElementById("myDIV");
    x.getElementsByTagName("P")[1].style.backgroundColor = "red";
}
</script>
```

getElementByTagName() Method

- Get all elements in the document

```
var x = document.getElementsByTagName("*");
```

- Find out how many <li> elements there are in the document (using the length property of the NodeList object):

```
var x = document.getElementsByTagName("LI").length;
```

getElementByClassName() Method

- The getElementsByClassName() method returns a collection of an element's child elements with the specified class name, as a NodeList object.

```
<div id="myDIV">
  <p class="child">First p element with class="child" in a div (index 0).</p>
  <p class="child">Second p element with class="child" in a div (index 1).</p>
  <p class="child">Third p element with class="child" in a div (index 2).</p>
</div>

<p>Click the button to add a background color to the second p element with
class="child" inside the div element.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
    var x = document.getElementById("myDIV");
    x.getElementsByClassName("child")[1].style.backgroundColor = "red";
}
</script>
```

Math Object

- The JavaScript Math object allows you to perform mathematical tasks on numbers.

- Examples ::
- Math.round(x) returns the value of x rounded to its nearest integer

```
Math.round(4.7);  // 5
```

- Math.pow(x,y) returns the value of x to the power of y

```
Math.pow(8,2);  // 64
```

- Math.sqrt(x) returns the square root of x

```
Math.sqrt(64); /8
```

- Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments:

```
Math.min(0, 150, 30, 20, -8, -200);  // returns -200
Math.max(0, 150, 30, 20, -8, -200);  // returns 150
```

Regular Expression

- A regular expression is a sequence of characters that forms a search pattern.
- The search pattern can be used for text search and text replace operations.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Regular Expression

- In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().

- **The search() method** uses an expression to search for a match, and returns the position of the match.

- **The replace() method** returns a modified string where the pattern is replaced.

Regular Expression

- **String search() With a Regular Expression**

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
var str = "Visit W3Schools";
var n = str.search(/w3schools/i); // I is used to do case-insensitive search
```

Output :: 6

- **String replace() With a Regular Expression**

Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:

```
var str = "Visit Microsoft!";
  var res = str.replace(/microsoft/i, "W3Schools");
```

Output :: Visit W3Schools!

Regular Expression Modifier

Modifiers can be used to perform case-insensitive more global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Regular Expression Patterns

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x|y)	Find any of the alternatives separated with |

Metacharacters are characters with a special meaning:

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning or at the end of a word

Using the RegExp Object

- In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.

- **Using test()**
- The test() method is a RegExp expression method.
- It searches a string for a pattern, and returns true or false, depending on the result.

The following example searches a string for the character "e":

```
var patt = /e/;
patt.test("The best things in life are free!");
```

Result :: true

Regular Expression

- **Using exec()**
- The exec() method is a RegExp expression method.
- It searches a string for a specified pattern, and returns the found text.
- If no match is found, it returns *null*.

The following example searches a string for the character "e":

/e/.exec("The best things in life are free!");

Since there is an "e" in the string, the output of the code above will be:

Result e

Best Practises

- **Use === Instead of ==**

when working with == and *!*=, you'll run into issues when working with different types. In these cases, they'll try to coerce the values, unsuccessfully.

- **Place Scripts at the Bottom of Your Page**

This tip has already been recommended in the previous article in this series. As it's highly appropriate though, I'll paste in the information.

Remember -- the primary goal is to make the page load as quickly as possible for the user. When loading a script, the browser can't continue on until the entire file has been loaded. Thus, the user will have to wait longer before noticing any progress.

If you have JS files whose only purpose is to add functionality -- for example, after a button is clicked -- go ahead and place those files at the bottom, just before the closing body tag. This is absolutely a best practice

Best Practises

- **Declare Variables Outside of the For Statement**

When executing lengthy "for" statements, don't make the engine work any harder than it must. For example:

Bad

```
for(var i = 0; i < someArray.length; i++) {
   var container = document.getElementById('container');
   container.innerHtml += 'my number: ' + i;
   console.log(i);
}
```

Notice how we must determine the length of the array for each iteration, and how we traverse the dom to find the "container" element each time -- highly inefficient!

```
var container = document.getElementById('container');
for(var i = 0, len = someArray.length; i < len;  i++) {
   container.innerHtml += 'my number: ' + i;
   console.log(i);
}
```

Best Practises

- **Avoid Global Variables**
- Minimize the use of global variables.
- This includes all data types, objects, and functions.
- Global variables and functions can be overwritten by other scripts.
- Use local variables instead, and learn how to use closures
- **Example ::**

```
var name = 'Jeffrey';
var lastName = 'Way';
function doSomething() {...}
console.log(name); // Jeffrey -- or window.name

var DudeNameSpace = {
  name : 'Jeffrey',
  lastName : 'Way',
  doSomething : function() {...}
}
console.log(DudeNameSpace.name); // Jeffrey
```

Best Practises

- **Don't Pass a String to "SetInterval" or "SetTimeOut“**

```
setInterval(
"document.getElementById('container').innerHTML += 'My new number: ' + i", 3000
);
```

Not only is this code inefficient, but it also functions in the same way as the "eval" function would. Never pass a string to SetInterval and SetTimeOut. Instead, pass a function name.

```
setInterval(someFunction, 3000);
```

Best Practises

- **Don't Use new Object()**

Use {} instead of new Object()

Use "" instead of new String()

Use 0 instead of new Number()

Use false instead of new Boolean()

Use [] instead of new Array()

Use /()/ instead of new RegExp()

Use function (){} instead of new Function()

- Example

```
var x1 = {};           // new object
var x2 = "";           // new primitive string
var x3 = 0;            // new primitive number
var x4 = false;        // new primitive boolean
var x5 = [];           // new array object
var x6 = /()/;         // new regexp object
var x7 = function(){}; // new function object
```

Best Practises

- **Use {} Instead of New Object()**

There are multiple ways to create objects in JavaScript. Perhaps the more traditional method is to use the "new" constructor, like so:

```
var o = new Object();
o.name = 'Jeffrey';
o.lastName = 'Way';
o.someFunction = function() {
  console.log(this.name);
}
```

However, this method receives the "bad practice" stamp without actually being so.

```
var o = {
  name: 'Jeffrey',
  lastName = 'Way',
  someFunction : function() {
    console.log(this.name);
  }
};
```

Note that if you simply want to create an empty object, {} will do the trick.

Best Practises

- **Use [] Instead of New Array()**

The same applies for creating a new array.

Okay

```
var a = new Array();
a[0] = "Joe";
a[1] = 'Plumber';
```

Better

```
var a = ['Joe','Plumber'];
```

Best Practises

- **Long List of Variables? Omit the "Var" Keyword and Use Commas Instead**
- It is a good coding practice to initialize variables when you declare them.
- Give cleaner code
- Provide a single place to initialize variables
- Avoid undefined values

```
var someItem = 'some string';
var anotherItem = 'another string';
var oneMoreItem = 'one more string';
```

Better

```
var someItem = 'some string',
    anotherItem = 'another string',
    oneMoreItem = 'one more string';
```

Best Practises

- **Always, Always Use Semicolons**

Technically, most browsers will allow you to get away with omitting semi-colons.

```
var someItem = 'some string'
function doSomething() {
  return 'something'
}
```

Having said that, this is a very bad practice that can potentially lead to much bigger, and harder to find, issues.

Better

```
var someItem = 'some string';
function doSomething() {
  return 'something';
}
```

Best Practises

- **Beware of Automatic Type Conversions**

JavaScript is loosely typed. A variable can contain different data types, and a variable can change its data type:

Example

```
var x = "Hello";     // typeof x is a string
x = 5;               // changes typeof x to a number
```

- **Avoid Using eval()**

The eval() function is used to run text as code. In almost all cases, it should not be necessary to use it.

Because it allows arbitrary code to be run, it also represents a security problem.

Mistakes

- **Expecting Loose Comparison**

In regular comparison, data type does not matter. This if statement returns true:

```
var x = 10;
var y = "10";
if (x == y)
```

- In strict comparison, data type does matter. This if statement returns false:

```
var x = 10;
var y = "10";
if (x === y)
```

Mistakes

- **It is a common mistake to forget that switch statements uses strict comparison:**

This case switch will display an alert:

```
var x = 10;
switch(x) {
    case 10: alert("Hello");
}
```

This case switch will not display an alert:

```
var x = 10;
switch(x) {
    case "10": alert("Hello");
}
```

Mistakes

- **Confusing Addition & Concatenation**
- **Addition** is about adding **numbers** and **Concatenation** is about adding **strings**.
- In JavaScript both operations use the same + operator.
- Because of this, adding a number as a number will produce a different result from adding a number as a string:

When adding two variables, it can be difficult to anticipate the result:

```
var x = 10;
var y = 5;
var z = x + y;          // the result in z is 15
var x = 10;
var y = "5";
var z = x + y;   // the result in z is "105"
```

Mistakes

- **Breaking a JavaScript String**
- JavaScript will allow you to break a statement into two lines:

Example 1

```
var x =
"Hello World!";
```

But, breaking a statement in the middle of a string will not work:

Example 2

```
var x = "Hello
World!";
```

You must use a "backslash" if you must break a statement in a string:

Example 3

```
var x = "Hello \
World!";
```

Mistakes

- **Misplacing Semicolon**

Because of a misplaced semicolon, this code block will execute regardless of the value of x:

```
if (x == 19);
{
   // code block
}
```

- **Ending an Array Definition with a Comma**

Incorrect:

```
points = [40, 100, 1, 5, 25, 10,];
```

Some JSON and JavaScript engines will fail, or behave unexpectedly.

Correct:

```
points = [40, 100, 1, 5, 25, 10];
```

Questions

Thank You!

Virtuoso InfoTech Pvt. Ltd.
4th Floor, Victory Landmark Opp. D Mart
Baner Road, Pune – 411045, Maharashtra, India

020 – 6050 1318
support@virtuoositech.com

www.virtuosoitech.com