JAVA SCRIPT

The introduction of JavaScript into web pages immediately ran into the Web's primary language, HTML. As the component of its original work on JavaScript, Netscape tried to find out how to make JavaScript coexist in HTML pages without causing any break on those pages, which is rendering in other browsers. Through trial and error along with controversy, several decisions were finally made and agreed upon to bring universal scripting support for the Web. Most of the work done in these early days of the Web has endured and become official in the HTML specification.

The primary technique of inserting JavaScript into an HTML page is through the <script> element. Netscape created this <script> element and first implemented in the browser - Netscape Navigator version 2. Then it was later added to the formal HTML specification.
Six attributes are provided by the <script> element. These are:

Attribute	Description
async	Which indicates that the script ought to begin downloading immediately.
charset	Set the character set of the code particularizes using the src attribute.
defer	This indicates that the execution of the script can safely be deferred until after the document's content has been fully parsed and displayed.
language	Which indicates that the scripting language being used by the code.
src	Which indicates that an external file that holds the code to be executed.
type	is used to replace language; indicates the content type (also called MIME type) of the scripting language being used.

JAVASCRIPT ATTRIBUTE:
HTML5 bring the async attribute for <script> elements. The async attribute is alike to defer (which you have read earlier) in that it transforms the way a script is processed. Further similar to the defer attribute, async applies only to external scripts and signals which the browser begins for downloading the file immediately. Similar to that of defer, scripts marked as async does not guarantee that execution will be in the order in which they are specified.
Let us take a code snippet:

```
<!DOCTYPE html><html><head>
    <title>Example HTML Page</title>
    <script type="text/javascript" async src="script.js"></script>
    <script type="text/javascript" async src="script1.js"></script> </head><body>
    <!— body content here --></body></html>
```

In the above code, the second script file might start execution before the first, and so there must be no dependencies among the two. The point of specifying an "async" script is to designate that the page need not have to wait for the script to which has to be downloaded and then executed before continuing to load it also need not wait for another script to load and execute before it can do the same.

KEYWORDS:

Keywords are reserved words in JavaScript that you cannot use to indicate variable labels or function names. There are a total of 63 keywords that JavaScript provides to programmers. All of them are shown in the below-mentioned table:
You can't use a keyword as an identifier in your JavaScript programs, its reserved words, and used to perform an internal operation.

abstract	arguments	boolean	break
byte	case	catch	char
const	continue	debugger	default
delete	do	double	else
eval	false	final	finally
float	for	function	goto
if	implements	in	instanceof
int	interface	let	long
native	new	null	package
private	protected	public	return
short	static	switch	synchronized
this	throw	throws	transient
true	try	typeof	var
void	volatile	while	with
yield			

class	enum	export	extends
import	super		

COMMENTS:
There are two ways to include comments in JavaScript.
ECMA-Script standardizes for JavaScript the C-style comments for both single-line comments as well as block comments. The single-line comment starts with two forward-slash characters, such as:
// comment applied for one line
A JavaScript block comment begins with a forward slash and asterisk (/ *) and ends with the opposite (* /), as in the example below:

```
/*

* This is the way of using multiline

* Comment...

*/
```

It is to be noted that even though the second and third lines contain an asterisk, these are not necessary and are included in the above code for clear readability.

```
<script>//This is a single line comment./*JavaScript program for Comments.*/
document.write("Comment on my JavaScript code!");</script>
```

Comment on my JavaScript code!

FUNCTIONS:

A JavaScript function is a set of reusable code that can be called anywhere within the program you execute; it eliminates the need to write the same code over and over again. Also, it helps you write the codes in a modular format. "Functions" has a lot of features that make it very popular. Another use of "functions" is that it divides a large program into a few small and manageable "functions" that make it easier for programmers to read and debug modules of code.
There are two ways to define a function. These are:
by function declaration and
by function expression.
The first, function declaration, has the following format:

```
function functionName(arg0, arg1, arg2 …. argN) {

//function body

}
```

The function's name is followed by the function keyword, and in this method, the function's name is assigned. Mozilla Firefox, Safari, Google Chrome, and Opera all feature a non-standard name property on functions revealing the assigned name. This value is at all times equivalent to the identifier, which immediately follows the function keyword:

```
//works only on Mozilla Firefox, Safari, Google Chrome and Opera Browsers

alert("Test");   // functionName
```

The second way to create a function is to use a function expression. Function expressions have many forms. The most common are as follows:

```
var function_Name = function(arg0, arg1, arg2…. argN){

//function body

};
```

This prototype of function expression looks similar to a normal variable assignment. A function is produced and assigned to the variable function_Name. The newly created function is considered to be an anonymous function since it has no identifier after the keyword 'function'. Anonymous functions can also be sometimes termed as lambda functions. This means the name property is an empty string.
Function expressions act like other expressions and hence must be assigned before usage. The following causes an error:

```
sayHello();        //error - function doesn't exist yet

var sayHello = function(){

alert("Hello Test!");

};
```

Before using Function definition, programmers need to define the functions. The most ordinary way of defining a function in JavaScript is by using the 'function' keyword, followed by a unique name for the function then, a list of parameters (that might or might not be empty), and a statement block bounded by curly braces.
The basic syntax is given as follows:

```
<script type = "text/javascript">
<!--
function func_name(list-of-parameters)
{
//Set of statements
}
//-->
</script>
```

In the below-mentioned program, the function is being called using a button. Pressing the button will call the function, and all the statements within the function block will get be executed.

```
<html>
<head>

<script type="text/javascript">
function print_Hello()
{
document.write ("Hi Karlos");
}
</script>

</head>
<body>
<p>Press the button for calling the user-defined function</p>
<form>
<input type="button" onclick="print_Hello()" value="CLICK ME">
</form>
<p> Try using different sentences and P tag is used to give paragraph. . . . </p>
</body>
</html>
```

JAVA SCRIPT OBJECTS:

The simplest way to create a custom object is to create a new instance of the object and add properties and methods to it, as in the example mentioned below:

```
var person = new Object();
person.name = "Karlos";
person.age = 23;
person.job = "Network Engineer";
person.say_Name = function(){
alert (this.name);};
```

This example creates an object called the person that has three properties which are: name, age, and job, and one method (say_Name()). The say_Name() method is used to display the value of this.name, which resolves to person.name.

The previous example can be rewritten using literal object notation as follows:

```
var person = {
name: "Karlos",
age: 23,
job: "Network Engineer",
say_Name: function(){
alert(this.name);}};
```

The person object in the above example is equivalent to the person object in the prior example mentioned earlier, with all those same properties and methods added. These properties are all created with certain characteristics that define their behavior in JavaScript.

Since JavaScript is an object-oriented programming language and so a programming language can be called object-oriented when it provides programmers with at least four basic capabilities to develop:

Encapsulation: It is the capability for storing related information, whether data or methods, mutually in a single object.

Aggregation: It is the ability to store one object inside another.

Inheritance: A class can depend upon another class or number of classes and inherit their variables and methods for some specific use.

Polymorphism: It is the potential of the concept of OOP for writing one function or method which works in a variety of different ways.

Objects are composed of attributes, and when an attribute contains a function, it is considered to be a method of the object else; the attribute is considered a property.

Object properties can be any of the three basic data types or any of the abstract data types. Object properties are variables which are used within the object's methods, but as well can be globally visible variables which are used throughout the page.

The syntax for including any property to an object is:
Obj_Name . obj_Property = property_Value;
Example:
var obj1 = project.title;

BOM:
At the heart of the BOM is the window object that signifies an instance of the browser. The window object serves a dual purpose in browsers and acts as the JavaScript interface between the browser window and the ECMA-Script Global object. This ultimately means that every object, variable, and function defined in a web page makes use of the window as its Global object and has the right of entry to the methods like parse-Int().
Since the window object gets doubles as the ECMA-Script Global object, every other variable and function declared globally turned into properties and methods of the window object. Let us take the example given below:

```
var age = 26;function say_Age(){
    alert(this.age);}

alert (window.age);     //26
say_Age();       //26
window . say_Age();     //26
```

In the above example, a variable named 'age' and a function named 'say_Age()' is classified in the global scope that automatically places them on the window object. So, the variable age is also accessible as window .age, and the function say_Age() is also accessible via window.sayAge().
When a page contains frames, each frame has its window object, which is stored in the frames collection. Within that frames collection, the window objects get indexed both by number (ranging from 0 and going in the direction of left to right and then row by row) and by the name of the frame. Each window object has its name property that contains the name of the frame. Let us the following code snippet -

```
<html><head><title>Frameset Example</title></head><frameset rows="150,*">
 <frame src="frame1.html" name="top_Frame">
 <frameset cols="60%,60%">
 <frame src="another_frame.html" name="left_Frame">
 <frame src="yetAnother_frame.html" name="right_Frame">
 </frameset></frameset><noframes></noframes></html>
```

The above code snippet generates a frameset having one frame across the top and two frames below. Also, the top frame can be referenced by window.frames[0] or window.frames ["top_Frame"]. Though, you would most likely use the top object instead of a window to refer to these frames (creating its top .frames[0], for instance). The top object always points to the very top (outermost) frame, which is the browser window itself.
Shaping the size of a window cross-browser is not straight-forward in JavaScript. Internet Explorer 9+, Mozilla Firefox, Safari Browser, Opera, and Google Chrome - all these mentioned browsers supply four properties to deal with window size: innerWidth, innerHeight, outerWidth, and outerHeight.

```
var pageWidth = window.innerWidth, pageHeight = window. innerHeight;
if (typeof pageWidth != "number"){
 if (document. compatMode == "CSS1Compat"){
    pageWidth = document. documentElement . clientWidth;
    pageHeight = document. documentElement . clientHeight;
 } else {
    pageWidth = document . body . clientWidth;
    pageHeight = document . body . clientHeight;
 }}
```