

Question 1- What is JavaScript?

Answer -

“**JavaScript** is a high-level, dynamic, untyped, and interpreted programming language. It has been standardized in the ECMAScript language specification. Alongside HTML and CSS, it is one of the three core technologies of World Wide Webcontent production; the majority of websites employ it and it is supported by all modern Web browsers without plug-ins. JavaScript is prototype-based with first-class functions, making it a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles. It has an API for working with text, arrays, dates and regular expressions, but does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded”.

Fundamentals Of JavaScript

1. There are no specific data types (e.g. :-var t) in JavaScript.
2. There are if, for, switch, while, do while, break, continue; similar to Java or C# in JavaScript.
3. document.write() is used to display an output in JavaScript.
4. There are some dialogues in JavaScript, which are as follows:
Alert -- OK
Confirm -- OK/CANCEL
Prompt -- Input
5. Function: There is a 'function' keyword, which is used for function in JavaScript.

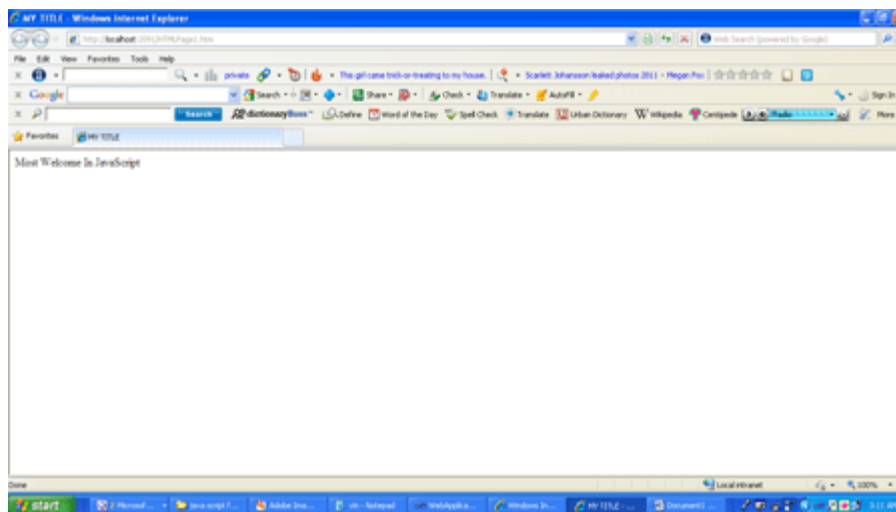
Some Examples of JavaScript -

Simple program of JavaScript.

```
• <html>  
•  
• <head>  
•     <title>MY TITLE</title>  
• </head>
```

-
- <body>
- <script type="text/javascript">
- document.write("Most Welcome in JavaScript")
- </script>
- </body>
-
- </html>

Output



For more details visit the following link,

- [Introducing JavaScript](#)

Question 2 - Explain Frames in JavaScript?

Answer -

Frames allow you to divide the page into several rectangular areas and to display a separate document in each rectangle. Each of those rectangles are called a "frame".

Here is an example:

- <FRAMESET ROWS="75%, *" COLS="*, 40%">
- <FRAME SRC="framea.html">
- <FRAME SRC="frameb.html">
- <FRAME SRC="framec.html">
- <FRAME SRC="framed.html">

- `<NOFRAMES>`
- `<H1>No Frames? No Problem!</H1>`

Take a look:

- `no-frames`

The version is:

- `</NOFRAMES>`
- `</FRAMESET>`
- `<FRAMESET ...>` is used instead of the `<BODY ...>` tag. The frameset file has no content, which appears on the page, so it has no need for `<BODY ...>`, which designates the content of the page. In fact, if you use `<BODY ...>` (except inside `<NOFRAMES>`), the frames will not appear. Tags in `<HEAD>`, including `<TITLE>`, still have their intended effects.

- Rows and columns are described by a list of widths or heights. For example `COLS="25%, *, 40%"` says that there will be three columns. The first column takes up 25% of the width of the page, the third column takes up 40% of the width of the page and the asterisk ("*") means "whatever is left over". See `COLS` and `ROWS` for more details.

Nested Frames

- `<FRAMESET ROWS="15%, *">`
- `<FRAME SRC="titlebar.html" NAME=TITLE`
`SCROLLING=NO>`
- `<FRAMESET COLS="20%, *">`
- `<FRAME SRC="sidebar.html" NAME=SIDEBAR>`
- `<FRAME SRC="menu.html" NAME=RECIPES>`
- `</FRAMESET>`
- `<NOFRAMES>` No frames? No Problem! Take a
look at our
- `no-frames`
version. `</NOFRAMES>`
- `</FRAMESET>`

Targeting Frames

Each frame is given a name, using `<FRAME NAME="...">`. These names uniquely identify each frame. Using these names, the links in other frames can tell the Browser, which frame does the link target.

- `<FRAMESET COLS="20%, *">`

- `<FRAME SRC="sidebar.html" NAME=SIDEBAR>`
- `<FRAME SRC="main.html" NAME=MAIN>`
- `</FRAMESET>`
- To target one of these frames, the link should have a TARGET attribute set to the name of the frame, where the linked page should appear. Thus, for example, this code creates a link totfetacos.html and targets the link to the MAIN frame:
- `my link`

For more details, visit the following link:

- [Understanding Frames](#)

Question 3 - What are the JavaScript Operators?

Answer -

JavaScript addition operator (+) serves two main purposes in JavaScript. The first is to perform a simple addition on the numbers.

The second operation is to perform a string concatenation (combine the strings).

Addition

- `<script>`
- `var a = 3;`
- `var b = 2;`
- `var c = a + b;`
- `document.write(c);`
- `</script>`

5

JavaScript subtraction operator

JavaScript subtraction operator (-) also serves two purposes in your code. The first is to perform simple subtraction on the numbers (6 - 2). The second operation is to specify negative numeric values (-20).

Subtraction

- `<script>`
- `var a = 10;`

- `var b = 3;`
- `var c = a - b;`
- `document.write(c);`
- `</script>`

7

JavaScript multiplication operator

JavaScript multiplication operator (`*`) is used to multiply the numbers.

- `<script>`
- `var a = 4;`
- `var b = 3;`
- `var c = a * b;`
- `document.write(c);`
- `</script>`

12

JavaScript division operator

JavaScript division operator (`/`) is used to divide the numbers.

- `<script>`
- `var a = 8;`
- `var b = 4;`
- `var c = a / b;`
- `document.write(c);`
- `</script>`

2

JavaScript increment operator

JavaScript increment operator (`++`) is used to increase a number by 1.

Example 1:

- `<script>`
- `var i = 0;`
- `i++;`
- `document.write(i);`
- `</script>`

1

JavaScript decrement operator

JavaScript decrement operator (--) is used to decrease a number by 1.

Example 1:

```
• <script>
•     var i = 1;
•     i--;
•     document.write(i);
• </script>
```

0

For more details visit the following link,

- [JavaScript Operators](#)

Question 4 - Explain JavaScript Statements.

Answer -

JavaScript block statement -

JavaScript block statements are the curly braces ({ }), you see everywhere. They are used to establish the code, which is to be compartmentalized into a specific function or a statement.

The lines of the code inside of a block statement is often intended to represent that they are the part of the block statement. The following are some examples, showing the curly braces establishing the blocks of the compartmentalized code.

```
• <script>
•     if (3 < 5) {
•         // code for this block statement goes in here
•     }
•     for (var i = 0; i < 10; i++) {
•         // code for this block statement goes in here
•     }
•
•     function myFunction() {
•         // code for this block statement goes in here
•     }
• </script>
```

JavaScript break statement

JavaScript break statement is used to terminate a loop, switch or label the statement from further processing. Apply it, when you want to force one of those types of statements to stop the processing.

Example of terminating a loop using break

```
• <script>
•   for (i = 0; i < 20; i++) {
•       if (i >= 5) {
•           break;
•       }
•       document.write("Pass index " + i + " of the
•   loop<br />");
•   }
• </script>
```

Pass index 0 of the loop

Pass index 1 of the loop

Pass index 2 of the loop

Pass index 3 of the loop

Pass index 4 of the loop

JavaScript continue statement

JavaScript continue statement is used to bypass the specified iterations of a loop, so that the code in the loop statement does not execute for the specified iterations and moves on to the next.

```
• <script>
•   for (i = 0; i < 20; i++) {
•       if (i < 10) {
•           continue;
•       }
•       document.write("Pass index " + i + " of the
•   loop<br />");
•   }
• </script>
```

Pass index 10 of the loop
Pass index 11 of the loop
Pass index 12 of the loop
Pass index 13 of the loop
Pass index 14 of the loop
Pass index 15 of the loop
Pass index 16 of the loop
Pass index 17 of the loop
Pass index 18 of the loop
Pass index 19 of the loop

JavaScript do...while statement

JavaScript do...while statement is an inverted while statement. It will execute the code as long as the while condition returns a value of true.

```
• <script>
•   var i = 0;
•   do {
•       document.write("Loop index: " + i + "<br />");
•       i++;
•   }
•   while (i < 5);
• </script>
```

Pass index 0 of the loop
Pass index 1 of the loop
Pass index 2 of the loop
Pass index 3 of the loop
Pass index 4 of the loop

JavaScript for statement

JavaScript for statement is a loop mechanism, which will execute code as long as the condition evaluation continues to be true.

```
• <script>
•   for (var i = 0; i < 5; i++) {
•       document.write("Pass index " + i + " of the
•       loop<br />");
•   }
• </script>
```


Pass index 0 of the loop
Pass index 1 of the loop
Pass index 2 of the loop
Pass index 3 of the loop
Pass index 4 of the loop

For more details visit the following link,

- [JavaScript Statements](#)

Question 5: Explain JavaScript objects.

Answer- Objects are the data types of JavaScript. Each variable that you create or literal value that you work with; is an object, which has specific methods and properties, that you can access, when working with that type of data, which is one of the biggest aspects of understanding any modern scripting language.

JavaScript also allows a script author to create the custom objects to extend the language. JavaScript supports an automatic data type casting for the objects. It simply means that we do not have to explicitly define each variable's data type, as we create them unless our script demands it.

JavaScript Standard Objects

- *String Object*
For working with the string data in JavaScript.
- *Array Object*
For compartmentalized data processing.
- *Date Object*
For date and time programming.
- *Math Object*
For mathematical calculations in JavaScript.
- *Number Object*
For working with the numeric data in JavaScript.
- *RegExp Object*
For matching a text against a pattern.

- *Boolean Object*
For representing false and true values.
- *Function Object*
For calling the dormant segments of your code to run.
- *object Object*
Extend JavaScript by creating your own custom objects

<http://www.c-sharpcorner.com/UploadFile/eda428/javascript-building-blocks/>

Question 6: Explain Arrays in JavaScript.

Answer - An array is a collection of similar data types. All its values are stored in the index locations, which start in the range 0 to n-1.

Declaration of an Array

- `var myArray = [];`
- `var myArray = [value1, value2, value3, so on...];`
- `var myArray = new myArray(length_of_array);`

Let's understand this with the following examples.

EXAMPLE: 1

- `var myArray = [20, 30, 40, 50];`
- `for (var i = 0; i <= myArray.length - 1; i++) {`
- `document.write("The value at the index location " + i`
- `+ " is " + myArray[i] + "
");`
- `}`

The preceding code declares an array with 4 values. The for loop is used, which starts the index value "i" from 0 until the length of "myArray" and the index location is incremented by 1.

The write property of the document object displays the values at the index location, which is implemented, using myArray[i] and HTML Break is used, so that the value of each index location is displayed in a different line.

EXAMPLE: 2

- `var myevenNumbersArray = [];`
- `for (var i = 0; i <= 5; i++) {`

```

•     myevenNumbersArray[i] = i * 2;
• }
• for (var i = 0; i < myevenNumbersArray.length; i++) {
•     document.write(myevenNumbersArray[i] + "<br/>");
• }

```

The preceding code prints all the even numbers, stored in the array. This time, we are adding the values to the array dynamically. We have used a for loop to do this. At the very first, we have declared an array, whose size is not defined. Thus, it means that it can contain as many values, as we want. We have used a for loop starting from 0 and going to 5. The value of the index location is multiplied by 2, each time, the interpreter iterates through the loop. Thus, 5 values will be stored in the array. In other words 0, 2, 4, 6, 8 and 10.

For more details, visit the following link,

- [Arrays in JavaScript](#)

Question 7: What is screen object in JavaScript?

Answer: Screen object helps in getting the information of the user's screen. We can get the width, height, colorDepth etc. of the user's screen. This information will be helpful in setting the size of the images, page to be displayed in the Browser. We will look into the properties, available in the screen object by an example.

Create a sample ASP.NET Web Application and a button, label and add JavaScript code, given below:

```

• <html xmlns="http://www.w3.org/1999/xhtml">
•
• <head runat="server">
•     <title></title>
•     <script language="javascript" type="text/javascript">
•         function showScreenData() {
•
•             document.getElementById("lblScreenDetails").innerText =
•                 "Total Height: " + screen.height + " Total Width: " +
•                 screen.width + " Available Height: " + screen.availHeight
•                 + " Available Width: " + screen.availWidth + " Available
•                 Pixel Depth : " + screen.pixelDepth + " Available Color
•                 Depth : " + screen.colorDepth + " Buffer Depth : " +
•                 screen.bufferDepth;
•
•                 return false;
•         }
•     }
• </script>
• </head>
• <body>
•     <div>
•         <button type="button" value="Show Screen Data" />
•         <label id="lblScreenDetails" />
•     </div>
• </body>
• </html>

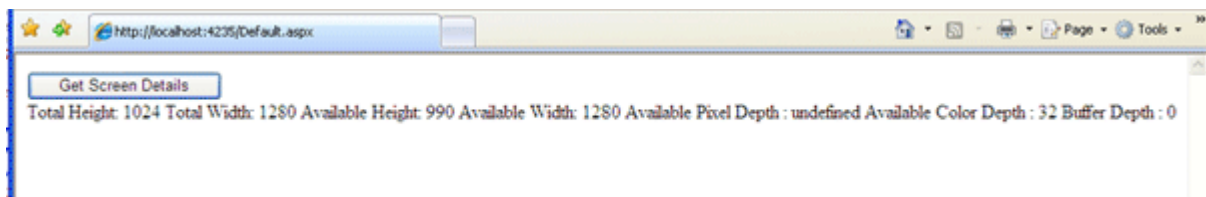
```

```

    }
</script>
</head>
<body>
    <form id="form1" runat="server">
        <div> </div>
        <asp:Button ID="btnScreenData" runat="server"
Text="Get Screen Details" OnClientClick="return
showScreenData()" /> <br />
        <asp:Label ID="lblScreenDetails" runat="server"
Text="Label"></asp:Label>
    </form>
</body>
</html>

```

Now, run the Application. It will show the properties of the screen like width, pixel depth etc., as shown below:



This way, we can access the user's screen properties for adjusting the images/page size accordingly.

By using the screen object, we can access the properties, given below, on the user's screen.

availHeight	Available Height
availWidth	Available Width
bufferDepth	-
colorDepth	-
deviceXDPI	-
deviceYDPI	-

height	Screen height
width	Screen width
logicalXDPI	-
logicalYDPI	-
fontSmoothingEnabled	-
updateInterval	-

For more details, visit the following link:

- [Introduction to screen object in JavaScript](#)

Question 8: What are Math Objects in JavaScript?

Answer - The Math object is used to perform simple and complex arithmetic operations.

The main properties and methods of Math Objects are given below:

Properties of Math Object

***E** - Holds the Euler's Number*

***LN2** - Natural Logarithm of 2.*

***LN10** - Natural Logarithm of 10.*

***LOG2E** - Base 2 Logarithm of E.*

***LOG10E** - Base 10 Logarithm of E.*

***PI** - Numerical Value of PI(22/7 or 3.142)*

***SQURT1_2** - Square root of $\frac{1}{2}$.*

***SQRT2** - Square root of 2.*

These are some main properties of Math objects in JavaScript. Now, let's learn about the methods in JavaScript. There is a long list of the methods for the Math objects and some of them are given below:

Methods of Math Object

`abs (x)` - Return the absolute value of x.
`acos (x)` - Return the arccosine (in radian) of x.
`asin (x)` - Return arcsine (in Radian) of x.
`atan (x)` - Returns the arctangent of x with numeric value b/w $-\pi/2$ to $+\pi/2$.
`atan2 (y,x)` - Returns the arctangent of quotient on dividing y and x.
`ceil (x)` - Rounds up x to the nearest bigger integer.
`cos (x)` - Return cosine value of x.
`exp (x)` - Returns the value of e^x .
`floor (x)` - Rounds up x to the nearest smaller integer.
`log (x)` - Returns the natural logarithmic value of x.
`max (x,y,z,.....n)` - Returns the highest number from the given list.
`min (x,y,z,.....n)` - Returns the smallest number from the given list.
`pow (x,y)` - Returns the x to the power of y.
`random ()` - Returns a Random number.
`round (x)` - Rounds up x to the nearest integer.
`sin (x)` - Return the sine value of x.
`sqrt (x)` - Returns the square root of x.
`tan (x)` - Returns the tangent value of x.

For more details, visit the following link:

- [Math Objects in JavaScript](#)

Question 9: What are the common Errors in JavaScript?

Answer - The common errors in JavaScript programming are the following:

1. Spelling and typing errors.
2. Missing brackets or quotation marks.
3. Mismatching quotation marks.
4. Using single equal sign instead of double equal sign in comparison.
5. Referencing objects that does not exist.
6. Using reserved keywords for the variable naming.
7. Using the wrong type of brackets.

These are the main causes of these errors.

In JavaScript, there are the following three types of errors:

1. Syntax Error
2. Runtime Error
3. Logic Error

Let's understand them one by one.

Syntax Error

Syntax errors are those errors, which occur, when we violate the rules, defined by JavaScript. These errors occur, when a Web Page that has been created is loaded.

Example: We are writing a Script to display the value of a variable but somewhere in the program, you forgot the closing parentheses, a syntax error will occur. The Syntax errors prevent JavaScript code from functioning. A Browser shows the error message with the specified line, when it encounters a JavaScript code error.

Runtime Errors

Runtime errors inform the user, that there is a problem with a script. Similar to the syntax errors, runtime errors are also displayed in a Browser. The nature of the error along with a line number is specified in an alert box, so you can easily search for the script. Runtime errors mostly occur due to improper use of the commands.

For instance, you will review runtime errors, if you reference a variable, which has not been defined. The following code snippet shows an example of a runtime error:

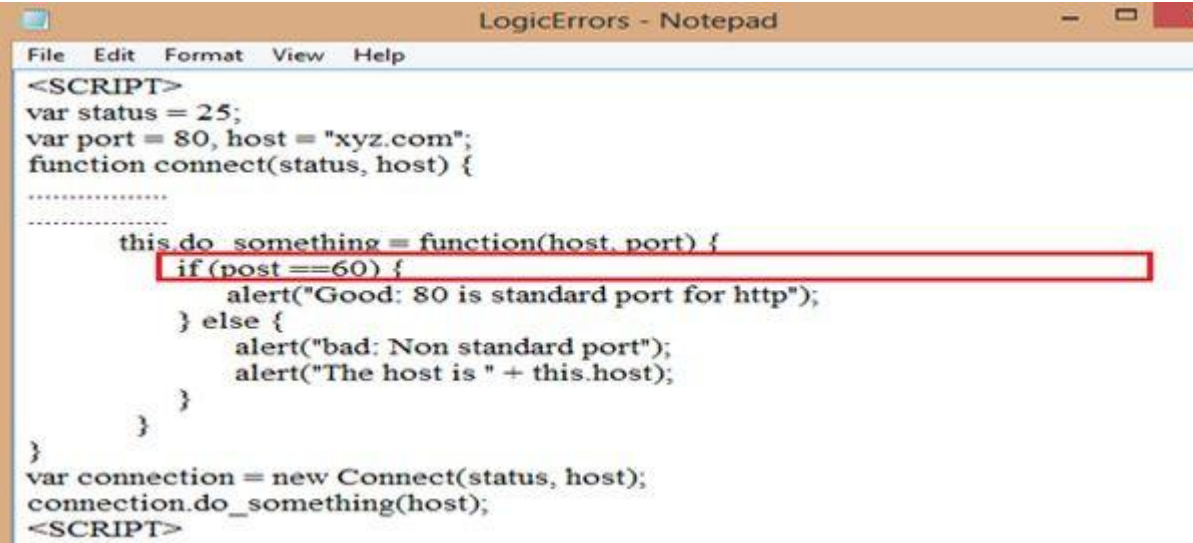
- `Sting = "C# Corner";`
- `document.write (sing);`

In the given code, we have declared a string variable, sting and assigned the value, "C# Corner", to it. Next, we displayed its value, but instead of writing "Sting", we wrote "sing", that is not declared. Therefore, the second line of the preceding code snippet generates a runtime error.

Logic Errors

Another type of common error is the logic, which occurs, when your script code does something different than the one suggested by the logic. Errors occur when the code is written incorrectly and you therefore do not get expected results. These errors are commonly known as bugs. Let's consider the following code:

Example



```
<SCRIPT>
var status = 25;
var port = 80, host = "xyz.com";
function connect(status, host) {
    .....
    this.do_something = function(host, port) {
        if(post == 60) {
            alert("Good: 80 is standard port for http");
        } else {
            alert("bad: Non standard port");
            alert("The host is " + this.host);
        }
    }
}
var connection = new Connect(status, host);
connection.do_something(host);
</SCRIPT>
```

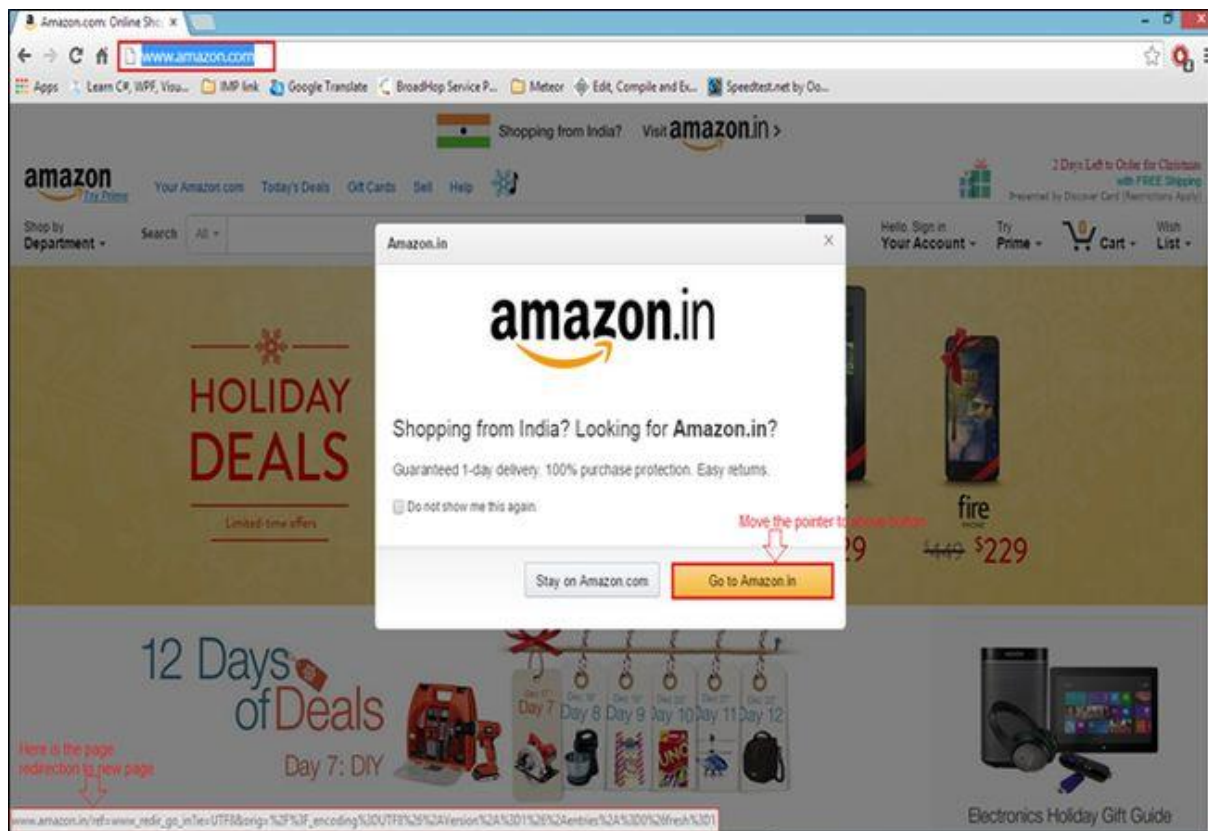
For more details, visit the following link:

- [Diagnosing Errors in JavaScript](#)

Question 10: Explain Page Re-direction in JavaScript.

Answer - Page redirection means moving to another location or the page, using JavaScript at the client side. Sometimes, you are visiting a specific Website but it

goes to another page, because it is internally redirected. If you visit the shopping Website: Amazon, it will show the image like:



It will show you a pop-up message “Shopping from India? Looking for Amazon.in?” with the two options, the first “Stay on Amazon.com” and the second “Go to Amazon.in” and when you click on the second option, the page will be redirected to Amazon.

There are multiple reasons to redirect from the original page, as follows:

1. One reason is that we just discussed in the previous example. The Browser gets your location and asks for the change of the page and the button click event page is redirected.
2. If you have created a new domain and you want to redirect all your visitors from the old to the new domain.
The following are the various page redirection methods to redirect from one page to another in JavaScript.

These methods are used for redirecting to another page; just add this code to the head section:

Using window.location.

Example:

```
• <script type="text/javascript">
•     <!--
•     window.location = "http://www.c-sharpcorner.com";
•     //-->
• </script>
```

Using window.location.href.

Example:

```
• <script type="text/javascript">
•     <!--
•     window.location.href =
•     "http://www.c-sharpcorner.com";
•     //-->
• </script>
```

Using window.location.assign.

Example

```
• <script type="text/javascript">
•     <!--
•     window.location.assign =
•     "http://www.c-sharpcorner.com";
•     //-->
• </script>
```

Using window.location.replace.

Example

```
• <script type="text/javascript">
•     <!--
•     window.location.replace =
•     "http://www.c-sharpcorner.com";
•     //-->
```

- `</script>`

Using window.open.

Example

```
• <html>
•
• <head>
•     <script type="text/javascript">
•         <!--
•             function WinOpen() {
•                 window.open("http://www.c-sharpcorner.com/",
• "OpenWindow", "status = 1, height = 450, width = 450,
• resizable = 0")
•             }
•             //-->
•         </script>
•     </head>
•
• <body> <input type="button" onClick="WinOpen()"
• value="WinOpen"> </body>
•
• </html>
```

For more details, visit the following link:

- [Page Re-direction in JavaScript: Day 6](#)

Question 11: Explain Scope and Events in JavaScript.

Answer - Scope in JavaScript

Scope defines the accessibility of a variable, objects, and function. It is nothing but the boundary line. There are only two types of scope, present in JavaScript as follows:

1. Local Scope in JavaScript
2. Global Scope in Script

Local Scope in JavaScript

It defines that something is only accessible on a limited scope. When you declare a variable within the function, it becomes local to that function. It's not accessible to the other functions or outside the function.

Example

```
• <!DOCTYPE html>
• <html>
• <title>Scope in JavaScript</title>
•
• <head></head>
•
• <body>
•     <script language='javascript'>
•         AuthorName();
•         document.write("</br>Outside the function</br>");
•         //can not access the author variable to outside
•         document.write("</br>Author is " + typeof
author);
•         document.write();
•
•         function AuthorName() {
•             //local variable declaration means local
scope
•             var author = "Jeetendra";
•             document.write("</br>Inside the function
</br>Author is " + author + "</br>");
•         }
•     </script>
• </body>
•
• </html>
```

Output

Inside the function
Author is Jeetendra

Outside the function
Global Scope in JavaScript

It can be accessible to the other functions, as it becomes global to all. You can access it within the function. It is defined anywhere in your JavaScript code.

Example

```
• <!DOCTYPE html>
• <html>
• <title>Scope in JavaScript</title>
•
• <head></head>
•
• <body>
•     <script language='javascript'>
•         document.write("Global Scope in
JavaScript<br>");
•         //global variabe declaration
•         var name = "krishna"; //it can be accessible to
all within JavaScript code
•         Name();
•
•         function Name() {
•             //access the test variable,
•             //it can be accessible because it is global
in scope
•             document.write("My Name is " + name);
•         }
•     </script>
• </body>
•
• </html>
```

Output

Global Scope in JavaScript

My Name is krishna

Events in JavaScript

The following figure shows the object hierarchy in JavaScript. All the objects have properties and methods. Some objects also have "events". Every element on a Web page has certain events, that can trigger invocation of the event handlers. The "event handler" is a command, that is used to specify the actions in response to an event. Attributes are inserted into HTML tags to define the events and event handlers.

Examples: The following are examples of events:

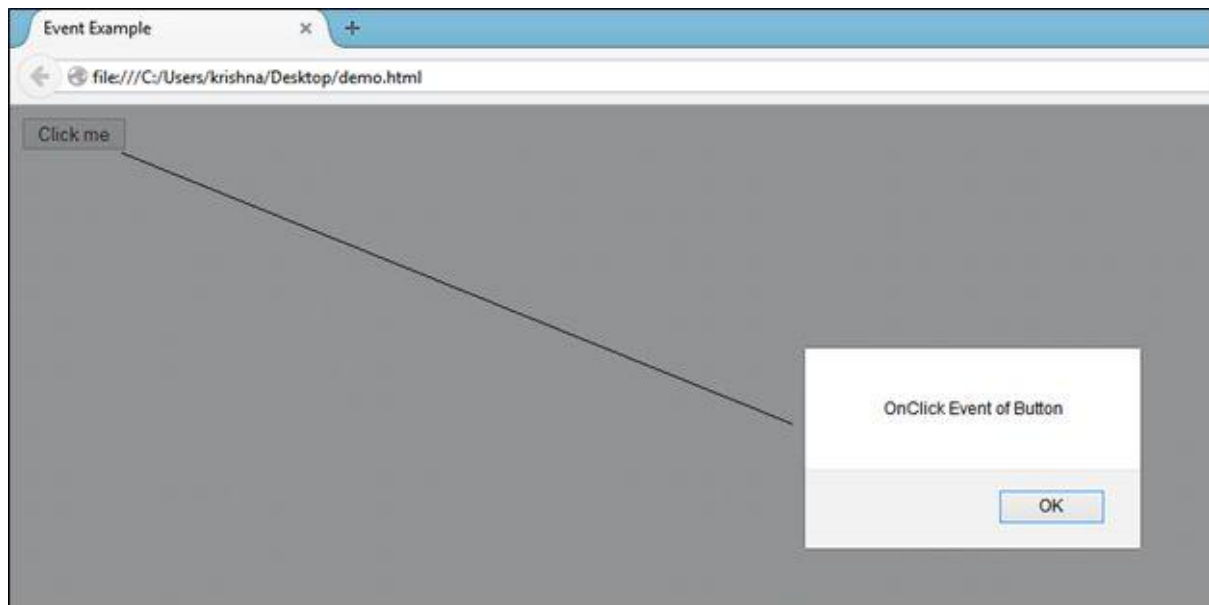
- A mouse click.
- A web page or an image loading.
- Moussing over a hot spot on the web page.
- Selecting an input box in an HTML form.
- Submitting an HTML form.
- A keystroke.

Example: OnClick event example.

```
• <!DOCTYPE html>
• <html>
•
• <head>Event Example</head>
•
• <body> <button onclick="clickme()">Click me</button>
•     <script language='javascript'>
•         function clickme() {
•             alert("OnClick Event of Button");
•         }
•     </script>
• </body>
•
• </html>
```

Run the preceding code, it will show you a button. When you click the button, it will pop up a message box, as shown in the following output:

Output



For more details, visit the following link:

- [Scope and Events in JavaScript: Day 4](#)

Question 12: What are the types of comments in JavaScript?

Answer - In JavaScript, the comments are used for skipping that statement from execution. Using the comments, you make the code more readable and understandable for anyone. Code functionality is clearer, using the comments. The following comments are used in JavaScript:

1. Single Line Comment
2. Multi-line comment

Single Line Comment

When you want to comment out a single statement, a single line comment is used. It starts with `"/"`. Using this, you can comment out an entire line. This line is ignored by JavaScript.

The following example uses a single line comment to explain the code:

```
• <!DOCTYPE html>
• <html>
• <title>JavaScript Comments</title>
•
• <head></head>
•
• <body>
•     <script language="javascript">
•         //Single line comment
•         //Addition of two numbers
•         var a = 25;
•         var b = 75;
•         var c = a + b; //addition of a and b is store in
c variable
•         document.write("Addition of " + a + " and " + b +
" is " + c);
•     </script>
• </body>
•
• </html>
```

Output: Addition of 25 and 75 is 100

Multi-line Comment

A multi-line comment is used with a group or the code block, which you want to comment out. A multi-line comment starts with `/*` and ends with `*/`. The code block between this is skipped by JavaScript.

In the following example, a group of statements are commented out, using the multi-line comments.

```
• <!DOCTYPE html>
• <html>
• <title>JavaScript Comments</title>
•
• <head></head>
•
• <body>
•     <script language="javascript">
•         /*
```



```

•         var a = 25;
•         var b = 75;
•         var c = a + b; //addition of a and b is store in
c variable
•         document.write("Addition of " + a + " and " + b
+ " is " + c);
•         */
•         function Print() {
•             document.write("Example of Multi-line
Comments in JavaScript");
•         }
•         Print();
•     </script>
• </body>
•
• </html>

```

Output: Example of the Multi-line Comments in JavaScript

For more details, visit the following link:

- [Comments and Objects in JavaScript: Day 3](#)

Question 13: Explain Exception Handling in JavaScript.

Answer - This is a new feature of JavaScript for handling the exceptions like in other programming languages, using try catch finally statements and the throw operator for handling the exceptions. Now, you can catch the runtime exception.

The try block is used always before a catch block, because in a try block, you provide your code to be executed. If there is no error in the code, the catch block is not executed but the final block is always executed. The following example shows an exception handling:

Examplethe

```

• <!DOCTYPE html>
• <html>
• <title>Article By Jeetendra</title>
•
• <head> </head>
•
• <body>

```

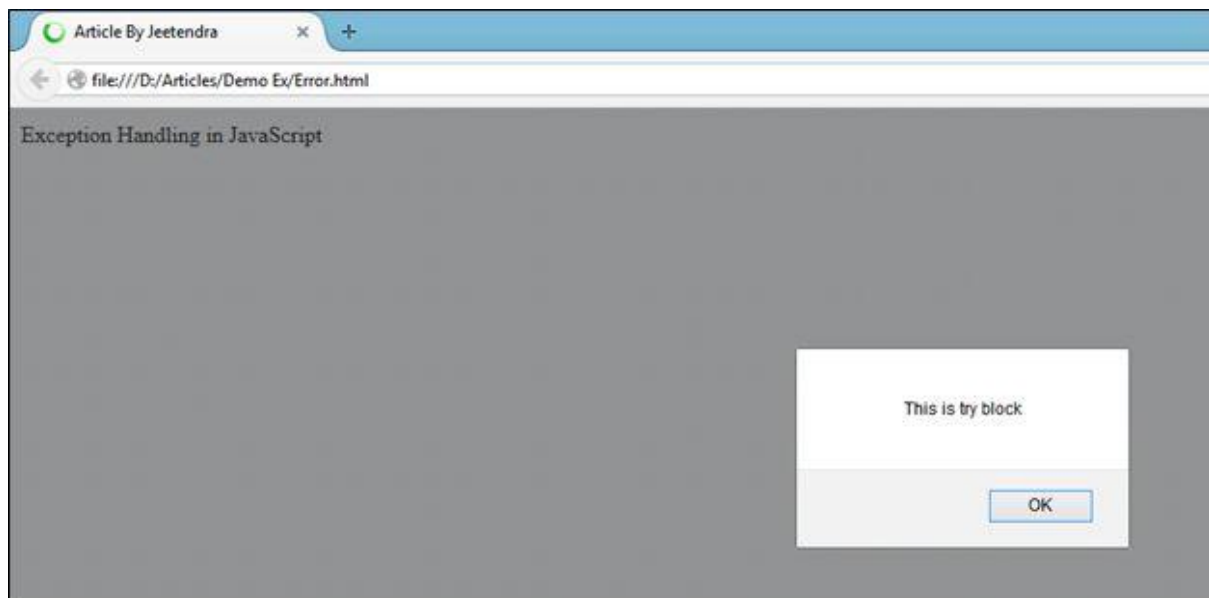
```

• <script type="text/javascript">
•     document.write("Exception Handling in
JavaScript<br>");
•
•     function ExceptHand() {
•         try {
•             alert("This is try block");
•             alert("Not present");
•         } catch (error) {
•             document.write(error);
•         }
•     }
•     ExceptHand();
• </script>
• </body>
•
• </html>

```

When you execute this program, it will first show an alert message box with the message "This is try block". When it executes the next statement, it is an error, because there is no alert message box available for this exception. It will print the proper exception message like the following output:

Output



For more details, visit the following link:

- [Errors and Exception Handling in JavaScript: Day 7](#)

Question 14: Explain the performance of JavaScript code.

Answer - There are ways to speed up your JavaScript code performance.

- **Reduction of activities in loops**

In our programming, we often use the loops for iteration.

For each iteration of the loop, every statement inside a loop is executed.

The statements or assignments which are to be searched can be placed outside the loop.

- **Reduction of DOM Access**

As compared to other JavaScript statements, accessing the HTML DOM is very slow.

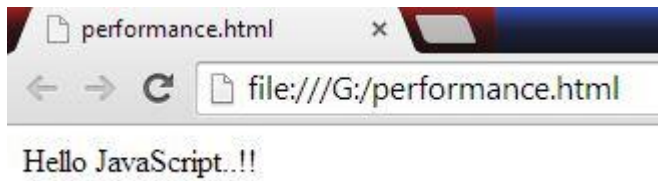
If you want to access the DOM element several times, access it once and use it as a local variable.

This is how you can access your DOM element easily as many times as you want.

Example

```
● <html>
●
● <body>
●   <p id="dom"></p>
●   <script>
●     var obj;
●     obj = document.getElementById("dom");
●     obj.innerHTML = "Hello JavaScript...!!";
●   </script>
● </body>
●
● </html>
```

- **Output**



- **Avoid Unnecessary variables**

Avoid creating the new variables that are not of use to save any value.

This will unnecessarily create a loading problem.

You can replace your code by optimizing the code.

Example

- `<p id="dom"></p>`
- `<script>`
- `var fullName = firstName + " " + lastName;`
- `fullName = document.getElementById("dom");`
- `fullName.innerHTML = fullName;`
- `</script>`



After reduction or optimization

- `<p id="dom"></p>`
- `<script>`
- `document.getElementById("dom").innerHTML =`
`firstName + " " + lastName;`
- `</script>`

For more details, visit the following link,

- [How to Speed up Your JavaScript Code Performance](#)

Question 15: Is JavaScript case sensitive?

Answer - Yes, JavaScript is a case sensitive scripting language. Variable names, keywords, methods, event handlers are all case sensitive.

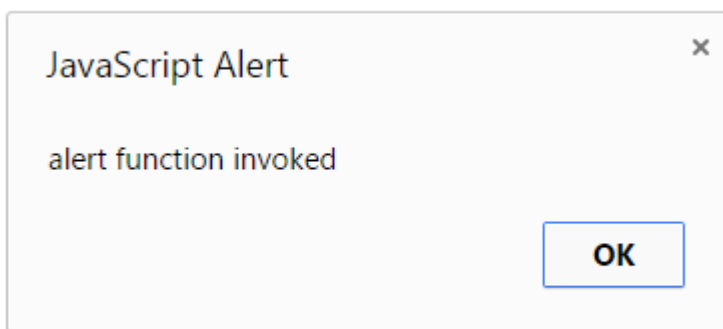
Example 1

In the previous part of this series, we saw how to use an alert function. We will be using the same alert function for this example.

```
• <html>
•
• <head>
•   <title></title>
• </head>
•
• <body>
•   <script type="text/javascript">
•       alert('alert function invoked');
•   </script>
• </body>
•
• </html>
```

We have a very pretty simple example. All we have is a script tag in the body section.

Inside the script tag, all we are doing is, we are calling the alert function. Thus, when we open this file in a Browser, the Browser will execute this JavaScript.



For more details, visit the following link:

- [Basics of JavaScript: Part 3](#)

Question 16: Describe JavaScript Anonymous Functions.

Answer - A function without a name is an anonymous function. We store these inside a variable name. Thus, the invocation happens, using the variable name.

Below is an example:

- `var sum = function(a,b){return a+b;};`
- `sum();`

Advantages of anonymous function

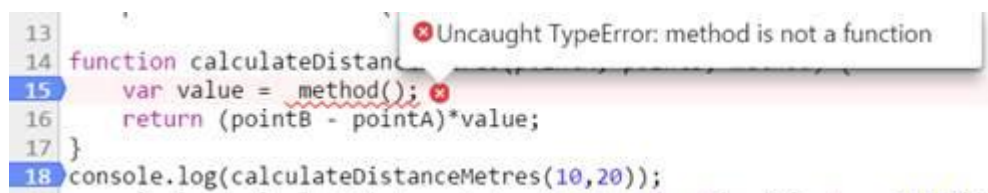
First advantage is that JavaScript allows you to pass an anonymous function as an object to the other functions. You don't need to explicitly define the function before passing it as a parameter to the other function. Example: below is a distance calculator between the two points. In method `calculateDistanceMetres`, we're passing an anonymous function.

- `function calculateDistanceMetres(pointA, pointB, method)`
- `{`
- `var value = method();`
- `return (pointB - pointA) * value;`
- `}`
- `console.log(calculateDistanceMetres(10, 20, function() {`
- `return 1000;`
- `}));`

Now, what if we don't pass an anonymous function, what will happen?

- `console.log (calculateDistanceMetres(10,20));`

It will throw an error because `method()` cannot execute any code.



In order to solve the above case, let us adjust our code, which will cover both the scenarios:

- `function` calculateDistanceMetres(pointA, pointB, method) {
- `if` (method) {
- `var` value = method();
- `return` (pointB - pointA) * value;
- } `else` {
- `return` (pointB - pointA);
- }
- }
- console.log(calculateDistanceMetres(10, 20)); //output 10
- console.log(calculateDistanceMetres(10, 20, `function`() {
- `return` 1000;
- })); //output 10000

Disadvantages of anonymous function

- If you have a large program, debugging is tough because you don't know the name of the function in the call stack. The call stack will show it as anonymous function.

Debug above program in Chrome & see Call Stack. You can notice an anonymous function, when the execution is at code `function () { return 1000; }`



- These functions cannot be reused.
- These functions cannot be unit tested easily. Therefore, you may need to refactor the code.

For more details, visit the following link:

- [Voice of a Developer: JavaScript Anonymous Functions - Part Twelve](#)

Question 17: Explain different types of JavaScript functions invocations.

Answer - Functions are the first class citizens in JavaScript. There are many ways to declare the functions in JavaScript. Let's understand each.

Function Declaration

The traditional way of defining a function is:

```
• function logMe() //function name logMe
• {
•     console.log('Log me in');
• }
• logMe();
```

Function hoisting

It is the process in which JavaScript runtime hoists all the functions declared, using the function declaration syntax at the top of JavaScript file. Look at the example, given below:

```
• function logMe() {
•     console.log('Log me in');
• }
• logMe();
•
• function logMe() {
•     console.log('Log me again');
• }
• logMe();
```

FUNCTION EXPRESSION

It's the second form of declaring a function.

```
• var log = function() {
•     console.log('Log me in');
• }
```

Anonymous function

It's a function without a name. You can assign a function to a variable. Another way of declaring an anonymous function is within an object literal. For example:

```
• var programmer = {
•     lang: 'Javascript',
•     browser: 'Chrome',
•     getFullSpecs: function() {
•         console.log(this.lang + ' running inside ' +
• this.browser);
•     }
• }
```


- }

Function() Constructor

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

- `var multiply = new Function("x", "y", "var z=x*y; return z;");`
- `multiply(10,20);`

Output: 200

For more details, visit the following link:

- [Voice of a Developer: JavaScript Functions Invocations - Part 11](#)

Question 20: What is Closure in JavaScript?

Answer - Basically a closure is a local variable, which will be executed after the specific function has returned. A Closure provides us a free environment, where the outer function can easily access the inner functions and inner variables without any scope restrictions.

Example

- `<html>`
- `</html>`
- `<head>`
- `<script type="text/javascript" language="JavaScript">`
- `function ExClosure(a, ref1) {`
- `var v1 = a;`
- `var ref = ref1;`
- `return function(v2) {`
- `v1 += v2;`
- `alert("Add:" + v1)`
- `alert("Reference Value:" + ref.cv);`
- `}`
- `}`
- `myclosure = ExClosure(10, {`
- `cv: 'My First Closure'`
- `});`
- `myclosure(5);`

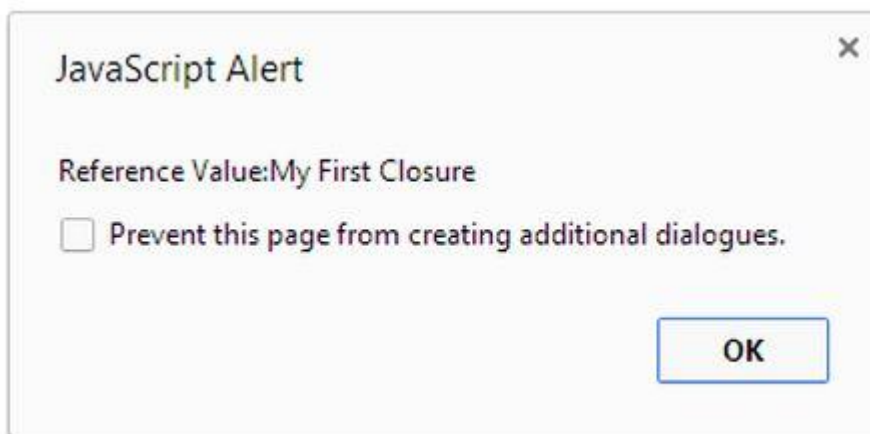
```

•     </script>
• </head>
•
• <body onload="ExClosure(b,ref) "> </body>
•
• </html>

```

In this example, when the ExClosure function is called, it returns a function. The function remembers the value of a (10) in the form of v1, it means the myclosure will add 10 together with 5 and return 15 as an alert and the next alert returns the Reference value (Reference Value: My First Closure).

The output will be:



For more details, visit the following link:

- [Closures in JavaScript](#)

Question 21: Explain Generic Function in JavaScript.

Answer - A generic function in JavaScript is similar to "String.IsNullOrEmpty" of C# to check whether the specified string is null or empty.

How often do we check a string for Null, Undefined & empty string in a JavaScript code? Here, is my very basic method StringHasValue to check, if a string is either null, undefined or if it's an empty string (String to be checked may contain the white-spaces is also considered). Although it's a very basic method but definitely it is going to save time, require less coding effort & it will be less error prone.

```
• var StringHasValue = function(strValue) {  
•     if ($.trim(strValue) != "" && $.trim(strValue) !=  
•     null && $.trim(strValue) != undefined) return true;  
•     else return false;  
• };
```

For more details, visit the following link:

- [Generic Function to Check Empty\Null string in JavaScript](#)

Question 22: Explain Break Statement in JavaScript.

Answer – Break statement allows you to break or exit a loop. When used inside a loop, the break statement stops executing the loop and causes the loop to be immediately exited. If the loop has statements after the break statement, the statements does not execute.

Write the following code:



```
<!DOCTYPE HTML>  
<HTML>  
  <HEAD>  
    <TITLE>Using the break statement</TITLE>  
  </HEAD>  
  <BODY>  
    <H1>Using the break statement.</H1>  
    <SCRIPT type="text/javascript">  
      var count=0;  
      while(count<10)  
      {  
        ++count;  
        if((count%5==0))  
          break;  
  
        else  
          document.write("count= "+count+"<BR/>");  
      }  
    </SCRIPT>  
  </BODY>  
</HTML>
```

Execute the script by opening the file in the Web Browser. If you are using Internet Explorer, click “Allow Blocked Content” to allow the script to execute and if you are using Mozilla Firefox, click allow “ActiveX Controls”.



Using the break statement.

```
count= 1  
count= 2  
count= 3  
count= 4
```

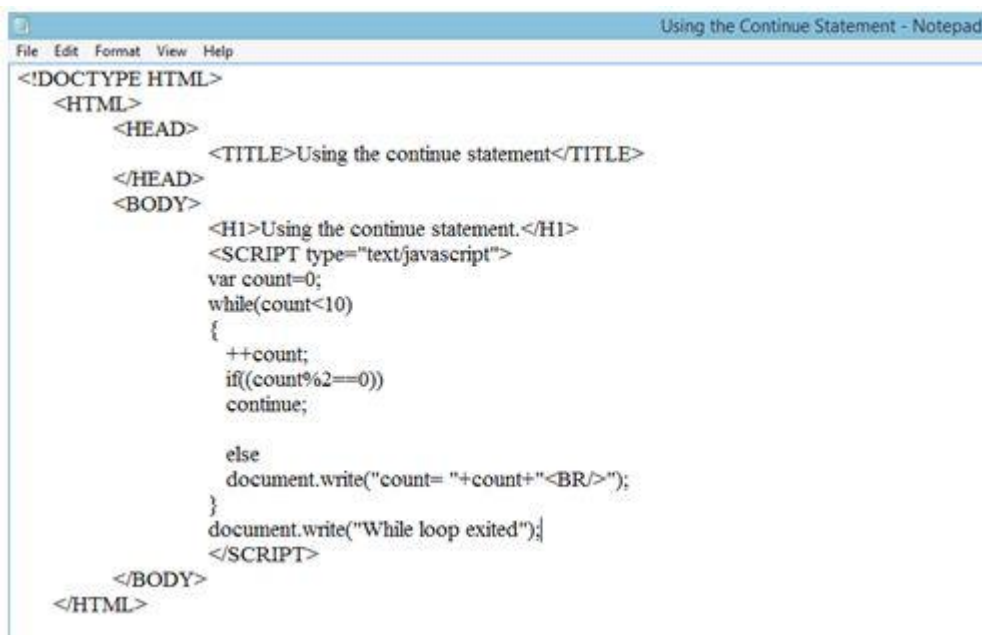
For more details, visit the following link:

- [Using Break Statement in JavaScript](#)

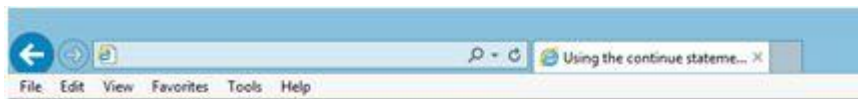
Question 23: Explain ContinueStatement in JavaScript.

Answer - Similar to the break statement, the continue statement is used to stop the execution of the loop. However, continue statement does not exit the loop; it executes the condition for the next iteration of the loop. If the loop has any statements after the continue statement, those statements are executed.

Write the following code:



Execute the script by opening the file in the Web Browser. If you are using Internet Explorer, click “Allow Blocked Content” to allow the script to execute and if you are using Mozilla Firefox, click allow “ActiveX Controls”.



Using the continue statement.

```
count= 1  
count= 3  
count= 5  
count= 7  
count= 9  
While loop exited
```

For more details, visit the following link:

- [Using the Continue Statement in JavaScript](#)

Question 23: What is InnerHTML Property in JavaScript?

Answer - InnerHTML property can be used to modify an HTML document. InnerHTML property exists in all types of major Browsers. When you use InnerHTML, you can change the page's content without refreshing the page. This can make your Website feel quicker and more responsive to user input.

Syntax

The syntax for using InnerHTML looks like this:

```
document.getElementById('ID of element').innerHTML = 'Data or content';
```

getElementById: getElementById refers to the HTML element using its ID.

Data or content: Data is the new content to go into the element.

JavaScript Code

To make it work, use the following JavaScript function code in your page:

```
• <script type="text/javascript">  
•     function ShowMessage() {  
•         document.getElementById('TextMessage').innerHTML  
•         = 'C-sharpCorner';  
•     }  
• </script>
```

Now, drag and drop a Button Control onto the form, so that you have:

- `<input type="button" onclick="ShowMessage()" value="Show Message" />`
- `<p id="TextMessage"></p>`

For more details, visit the following link:

- [Using GetElementById Function With InnerHTML Property in JavaScript](#)

Question 24: Explain Array Object in JavaScript.

Answer - An array object is used to store the multiple values in a single variable. Here, an array can hold various types of data types in a single array slot, which implies that an array can have a string, a number or an object in a single slot. It implies that an array can have a string, a number or an object in a single slot.

How to Create an Array Object:



Using the Array Constructor



Using the Array Literal notation

Using the Array Constructor

An empty array is used in the cases where we do know the exact number of the elements to be inserted in the array. We can create an empty array by using an array constructor.

Syntax

```
var mayarray = new array();
```

For more details, visit the following link:

- [Array Object in JavaScript](#)

Question 25: What is the procedure to save and get a value in cookies using JavaScript?

Answer - In the following function, we pass three parameters:

1. `c_name`: The name of the cookie (that you want to create).

2. *value*: The value, you want to save in the cookie.
3. *expiredays*: Expiry days of your cookie.

```
• function setCookie(c_name, value, expiredays) {  
•     var exdate = new Date();  
•     exdate.setDate(exdate.getDate() + expiredays);  
•     document.cookie = c_name + "=" + value + ";path=/" +  
•     ((expiredays == null) ? "" : ";expires=" +  
•     exdate.toGMTString());  
• }
```

Procedure to get the value from a cookie

In the following function, we pass one parameter:

Name: Name of the cookie, that you want to get the value of.

```
• function getCookie(name) {  
•     var dc = document.cookie;  
•     var prefix = name + "=";  
•     var begin = dc.indexOf("; " + prefix);  
•     if (begin == -1) {  
•         begin = dc.indexOf(prefix);  
•         if (begin != 0) return null;  
•     } else {  
•         begin += 2;  
•     }  
•     var end = document.cookie.indexOf(";", begin);  
•     if (end == -1) {  
•         end = dc.length;  
•     }  
•     return unescape(dc.substring(begin + prefix.length,  
•     end));  
• }
```

For more details, visit the following link:

- [How to Save and Get Values From Cookies Using JavaScript](#)

Question 26: What are Cookies in JavaScript?

Answer - Cookies are the variables stored on the visitor's (client's) computer. When the client's Browser requests a page, the cookies are obtained too. JavaScript can be used to create and retrieve cookie values.

Cookies are often the most efficient method of remembering and tracking the preferences, purchases, commissions and other information, required to improve the visitor and Website statistics.

Main components of a cookie:

Expire: The date on which the cookie will expire. If this is blank, the cookie will expire, when the visitor quits the Browser.

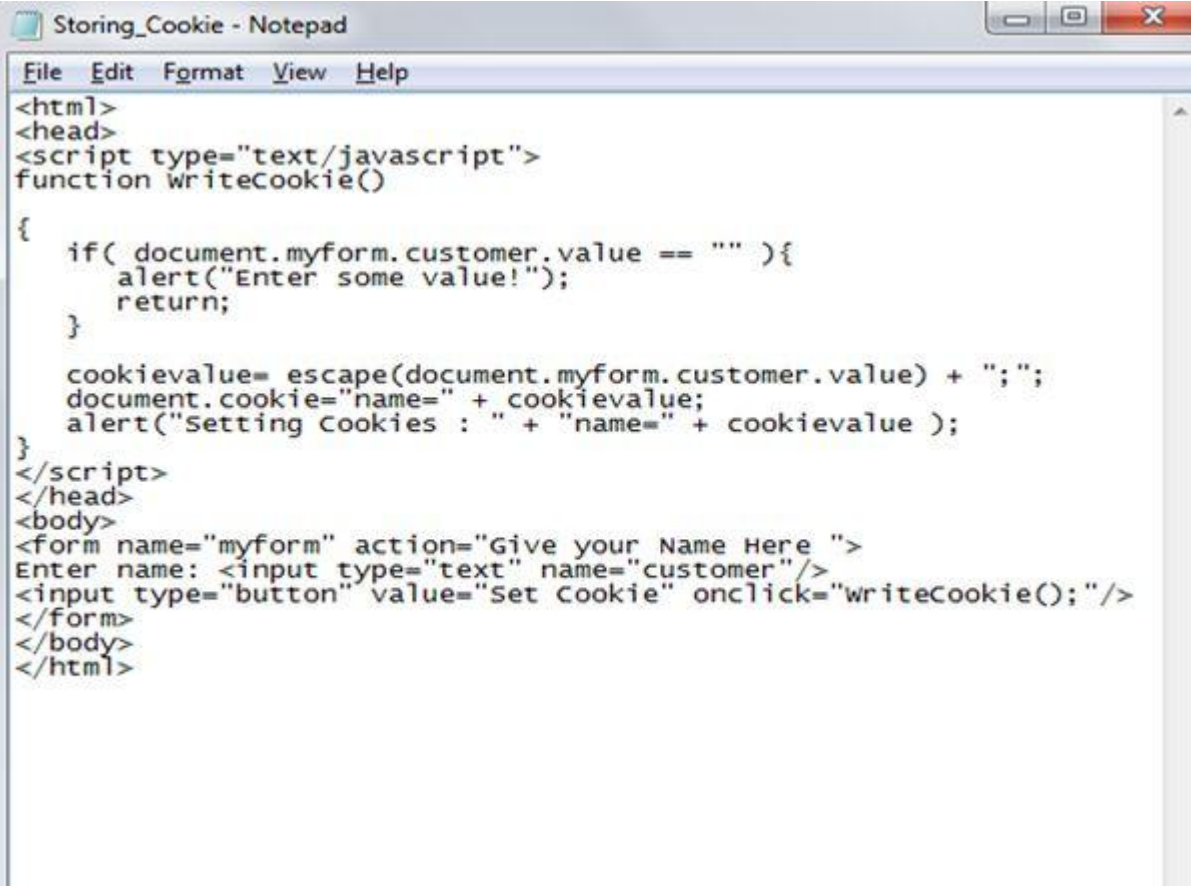
Domain: The domain is the name of our Website.

Path: The path to the directory or the Web page, that sets the cookie. This may be blank, if you want to retrieve the cookie from any directory or the page.

Secure: If this field contains the word "secure", 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 a key and value pairs.

Storing Cookies: The simplest way to create a cookie is to assign a string value to the "document.cookie" object, which looks like the syntax, mentioned above.



```
<html>
<head>
<script type="text/javascript">
function writeCookie()
{
    if( document.myform.customer.value == "" ){
        alert("Enter some value!");
        return;
    }

    cookievalue= escape(document.myform.customer.value) + ";";
    document.cookie="name=" + cookievalue;
    alert("Setting Cookies : " + "name=" + cookievalue );
}
</script>
</head>
<body>
<form name="myform" action="Give your Name Here ">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="writeCookie();"/>
</form>
</body>
</html>
```

The output of storing the cookie will be:



For more details, visit the following link:

- [Cookies in JavaScript](#)

Question 27: What are the Popup Boxes in JavaScript?

Answer – There are the following types of Popup Boxes in javascript, which are:

Alert Box

An alert box is often used, if you want to ensure the information comes through to the user and it displays some information to the user.

Function

```
• function showAlert() {  
•     alert("Hello C-sharpcorner");  
• }
```

LineBreak Box

A LineBreak alert box is often used to display the line breaks inside a popup box. Use a backslash, followed by the character n.

Function

```
• function ShowLinkBreak() {  
•     alert("Hello \n C-sharpcorner");  
• }
```

Confirmation Box

A JavaScript confirmation box is the same as an alert box. It is a way to prompt your users to explicitly confirm an action. It supplies the user with a choice; they can either press "OK" to confirm the popup's message or they can press "Cancel" and not agree to the popup's request.

Function

```
• function ShowConfirm() {  
•     var confirm = confirm("Are you sure you want to do  
•     that?");  
•     var status = document.getElementById("content");  
•     if (confirm == true) {  
•         status.innerHTML = "You confirmed, thanks";  
•     } else {  
•         status.innerHTML = "You cancelled the action";  
•     }  
• }
```

For more details, visit the following links:

- [Popup Boxes In JavaScript: Part 1](#)
- [Popup Boxes In JavaScript: Part 2](#)

Question 28: Discuss delete keyword in JavaScript.

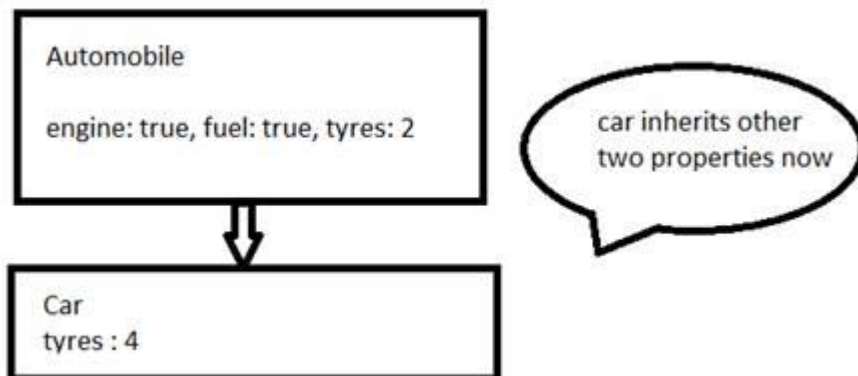
Answer - When we work on the large applications, we work with many objects. We are familiar with the problems like memory leak, performance of an application is not good and similar issues arise at a later stage. I always recommend that we know how much memory we are consuming in our application.

Ex- Chrome provides a quick way to look into the memory usage.

The main advantage is to delete the unwanted property from the object.

```
• var automobile = {engine: true, fuel: true, tyres:2 };  
• var car = Object.create (automobile); //car is instanceof  
• automobile  
• console.log (car.tyres); // 2 it inherits from parent  
• class  
• //as we know car has own tyres then we can create own  
• property of car object
```

- `car.tyres = 4; // will make own tyre property of car object`



- `//now we want to delete property tyres from car`
- `delete car.tyres ; //will return true`

For more details, visit the following link:

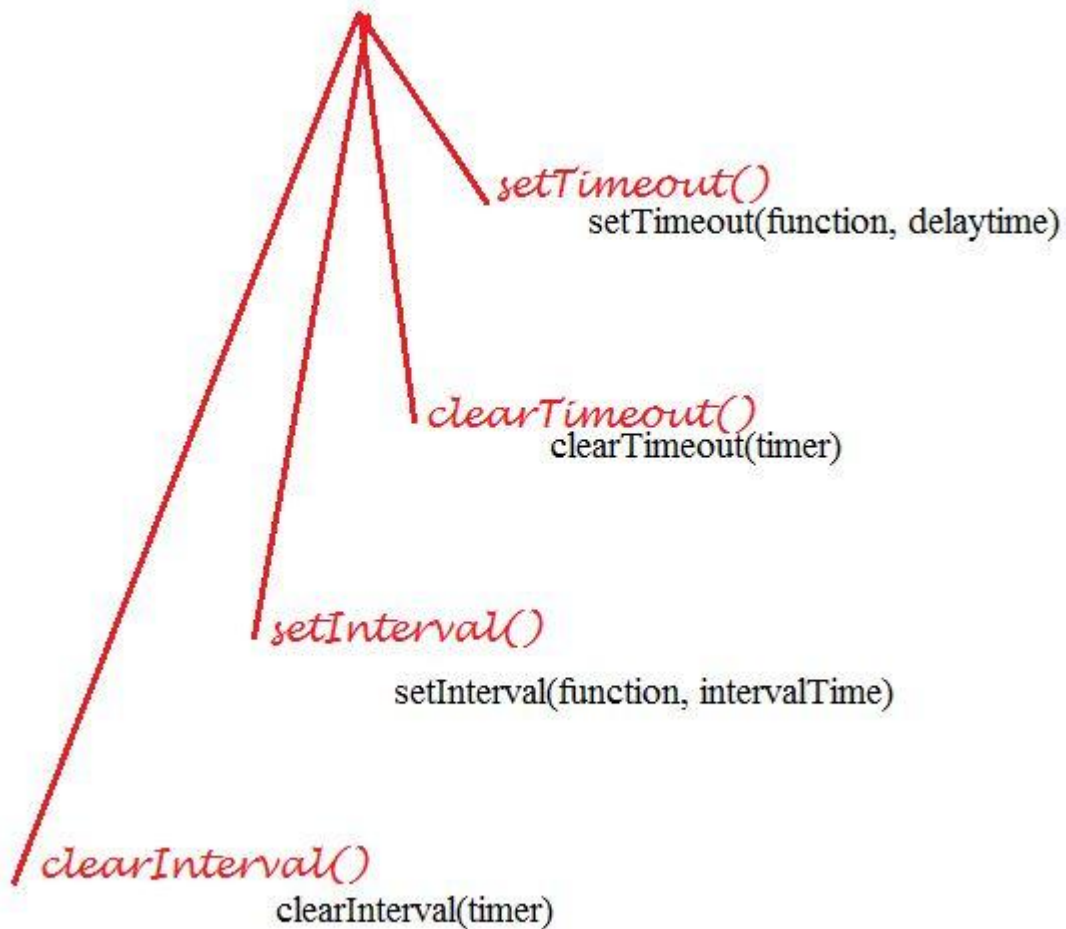
- [Voice of a Developer: Part Nine - JavaScript Useful Reserved Keywords](#)

Question 29: What is Timer in JavaScript?

Answer - In JavaScript, the timer is a very important feature, it allows us to execute a JavaScript function after a specified period, thereby making it possible to add a new dimension, time, to our Website. With the help of the timer, we can run a command at the specified intervals, run loops repeatedly at a predefined time and synchronize the multiple events in a particular time span.

There are various methods for using it as in the following:

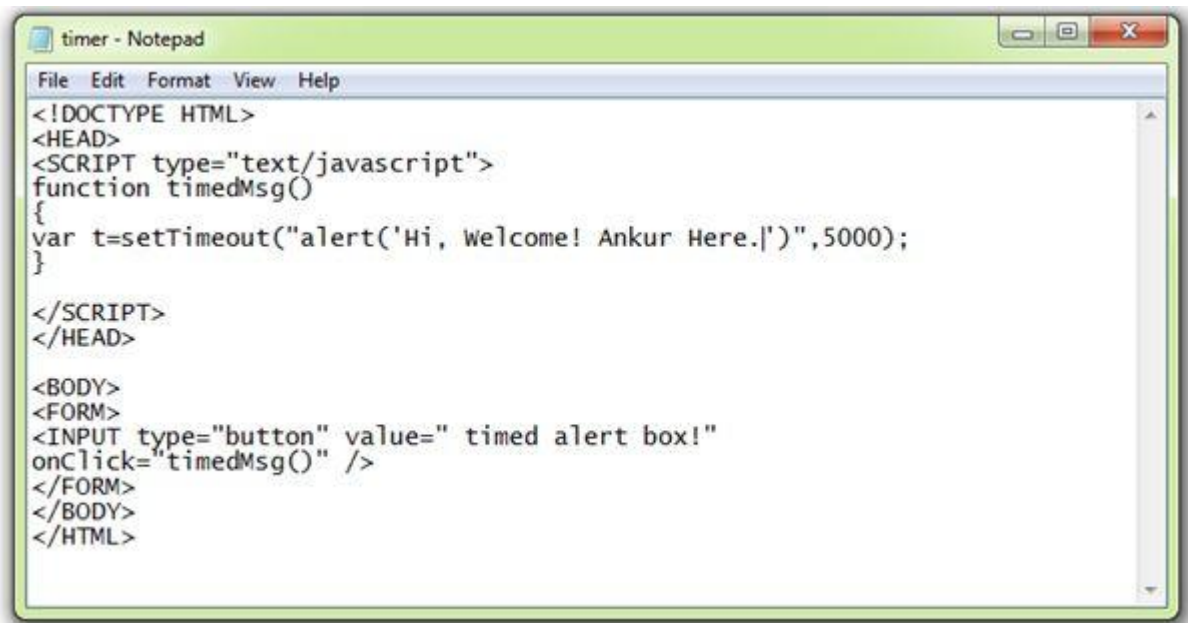
Timer Method in JavaScript



The setTimeout() method: Executes code at a specified interval. The syntax for this is:

setTimeout(function, delayTime)

Example: Let's try with an example. This method is useful for delaying the execution of a particular code. Let's write the code, as in the following:



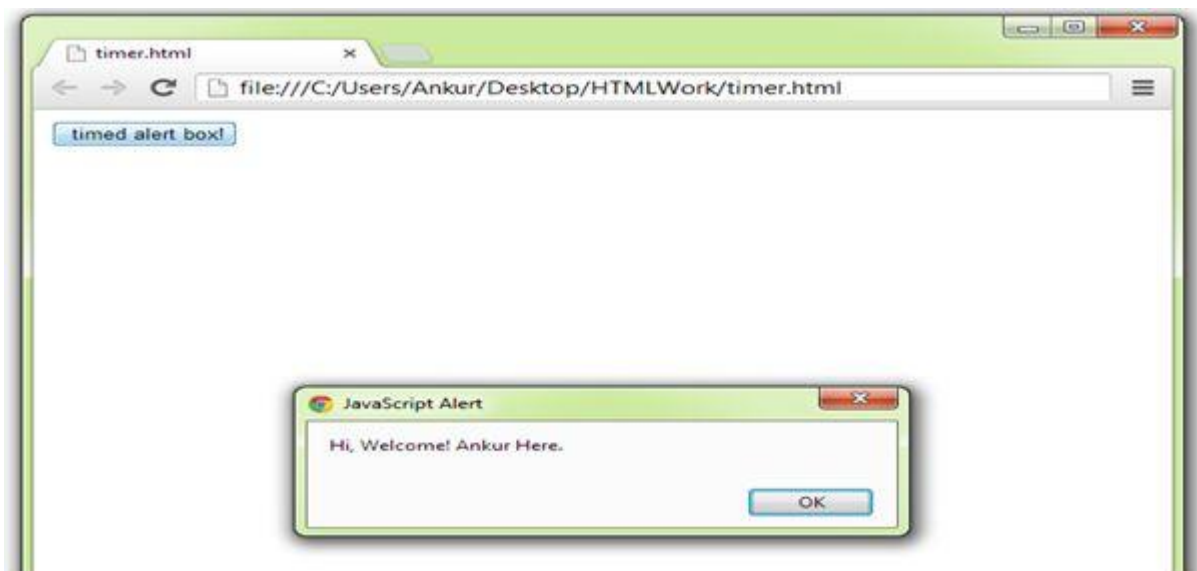
```
<!DOCTYPE HTML>
<HEAD>
<SCRIPT type="text/javascript">
function timedMsg()
{
var t=setTimeout("alert('Hi, Welcome! Ankur Here.')" ,5000);
}

</SCRIPT>
</HEAD>

<BODY>
<FORM>
<INPUT type="button" value=" timed alert box!"
onClick="timedMsg()" />
</FORM>
</BODY>
</HTML>
```

The setTimeout ("alert('Hi, Welcome! Ankur Here.')" , 5000) method creates a timed alert box. The first parameter of the setTimeout() method is a string, which contains a JavaScript statement, alert ('Hi, Welcome! Ankur Here.') and a second parameter , 5000 , specifies the time in milliseconds, after which the first parameter will execute.

The output of this code will be:



For more details, visit the following link:

- [Calling Function With Timer in JavaScript](#)

Question 30: Explain Dialog Boxes in JavaScript.

Answer - There are mostly three types of dialog boxes in JavaScript. They are used to either show a confirmation message, raise an error or show a warning message. You can get the input also from these dialog boxes. The following are the dialog boxes in JavaScript:

1. Alert Dialog Box
2. Prompt Dialog Box
3. Confirmation Dialog Box

Now, you can learn about JavaScript dialog boxes, one by one.

Alert Dialog Box

This dialog box is mostly used for the validation. It displays a message in the dialog box. This dialog box will block the Browser. You cannot do anything in the Browser page without clicking “OK” button of this dialog box and it is closed. It is used to display the error messages.

Example of an alert dialog box:

```
• <!DOCTYPE html>
• <html>
• <title>JavaScript Dialog Box</title>
•
• <head></head>
•
• <body>
•     <script language="javascript">
•         alert("Hello, From C# Corner");
•     </script>
• </body>
```

Prompt Dialog Box

It is the only dialog box that can get input from the user. This dialog box has the two buttons “OK” and “Cancel”. Using this dialog box, you can get input from the user to perform the operation you want on the input value. You can provide a default value for the prompt dialog box. The default value is “undefined”.

The following is an example of a prompt dialog box:

```
• <!DOCTYPE html>
• <html>
• <title>JavaScript Dialog Box</title>
•
• <head></head>
•
• <body>
•     <script language="javascript">
•         var test = prompt("Welcome To Coding World, Enter
Your Name:", "krishna");
•         document.write("Welcome to C# Corner - " + test);
•     </script>
• </body>
•
• </html>
```

Confirmation Dialog Box

This dialog box is mostly used for providing the confirmation for the user of specific actions. When you are exiting from a Window, it can request confirmation from the user, like “Do you really want to Exit?” It displays the two buttons on the dialog box: “OK” and “Cancel”. When you click “OK” button, it returns true, else it returns false.

The following is an example of a confirmation dialog box:

```
• <!DOCTYPE html>
• <html>
• <title>JavaScript Dialog Box</title>
•
• <head></head>
•
• <body>
•     <script language="javascript">
•         var t = confirm("Do you really want to Exit?");
•         if (t == true) {
•             document.write("Thanks for using");
•         }
•     </script>
• </body>
• </html>
```

```
•         } else {  
•         document.write("Welcome To C# Corner");  
•         }  
•     </script>  
• </body>  
•  
• </html>
```

For more details, visit the following link:

- [Dialog Boxes in JavaScript](#)

Question 31: What is a Variable in JavaScript?

Answer - They have a value stored in a program and:

- Variable names must begin with a letter.
- Variable names can also begin with \$ and _ (but we will not use it).
- Variable names are case sensitive (y and Y are the different variables).

There are many types of JavaScript variables, but for now, just think of the numbers and strings.

When you assign a text value to a variable, put double or single quotes around the value.

When you assign a numeric value to a variable, do not put quotes around the value. If you put quotes around a numeric value, it will be treated as a text.

There are two types of variables and they are:

1. Global Variable
2. Local Variable

Global Variable

A global variable is the one, which can be accessed, whether it is retrieving the value of or assigning a value to it, anywhere in an Application. Global variables are used for the variables that are required to be accessed throughout an Application, like totals, file paths, etc.

As a rule of thumb, it is best to avoid using too many global variables and instead use the properties for those items you want to access in other places, providing "get" and "set" accessors. Thus, the values cannot be changed accidentally.

Local variable

When a local variable is declared and accessed, it can only be of a specified class or method. Local variables are used for the variables which are only needed in a particular module/class/sub. You can have the local variables with the same name in the various functions, because the local variables are only recognized by the function in which they are declared

Basic Example

```
• <!doctype html>
• <html>
•
• <head>
•     <title>variable</title>
• </head>
•
• <body>
•     <script>
```

```

•      var x = 10; // It is a Global Variable
•      function add() {
•          var y = 20; //local variable
•          var some = 0; //Local variable
•          for (var i = x; i < y; i++) {
•              some = some + i;
•              document.write(some + "<br> ");
•          }
•      }
•      </script> <input type="button" value="submit"
•      onclick="add()" /> </body>
•
• </html>

```

Output



After clicking the "Submit" button, the output will be:

```

10
21
33
46
60
75
91
108
126
145

```

For more details, visit the following link:

- [Global and Local Variable in JavaScript](#)

Question 32: Explain Math Object In JavaScript.

Answer - JavaScript also provides "Math" object to perform mathematical operations. Math object has properties and methods, which helps to perform the mathematical tasks. Math object is not a constructor. All the methods and properties of Math are static, so they can be accessed, using Math as an object without creating one.

Math Properties

Math object has the following properties.

E	Returns Euler's constant, approximate value 2.718
LN2	Returns the natural logarithm of 2, approximate value 0.693
LN10	Returns the natural logarithm of 10, approximate value 2.302
LOG2E	Returns base 2 logarithm of E, approximate value 1.442
LOG10E	Returns base 10 logarithm of E, approximate value 0.434
PI	Returns PI, approximate value 3.14
SQRT1_2	Returns square root of $\frac{1}{2}$, approximate value 0.707
SQRT2	Returns square root of 2, approximate value 1.414

The properties, listed above, can be accessed as follows:

```
var pi=Math.PI;
```

Math Methods

Math object provides the following methods for the mathematical operations.

- abs():
- acos():
- asin():
- cos():
- tan():
- sin():
- ceil():
- floor():
- max():
- min():
- exp():
- log():
- pow():
- random ():

- `round ()`:
- `sqrt ()`:

For more details, visit the following link:

- [Math Object In JavaScript](#)

Question 33: What are the Event Handlers in JavaScript?

Answer - Event Handlers are considered as the triggers, which execute JavaScript when something happens, such as a click or move of your mouse over a link.

Here, I'll try to provide the proper syntax of some event handlers.

Some are as follows:

- `onClick`
- `onFocus`
- `onLoad`
- `onMouseOver`
- `onMouseOut`
- `onSelect`
- `onUnload`

onClick

`onClick` handlers execute something only, when the user clicks on the buttons, links and so on.

onLoad, onunLoad

The `onload` event handler is used to execute JavaScript after loading.

onMouseOver, onMouseOut

- `Welcome!`
- `Bye Bye!`

onFocus

- `<form>`
- `<input onFocus="this.value=''" type="text" value="Your email">`
- `</form>`

onSelect

onSelect event fires, when the target element has its text selected (highlighted by the user).

For more details, visit the following link:

- [Event Handlers in JavaScript](#)

Question 34: How can you create a Date Object and Compare Two Dates Using JavaScript?

Answer - The following describes the creation of Date objects:

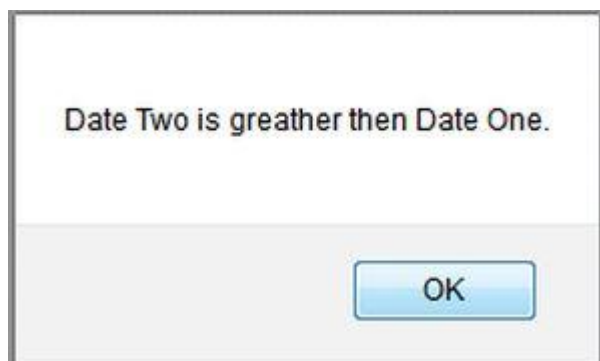
1. First, we need to create a date object.
2. The Date object is used to work with the date and time.
3. Date objects are created with the Date() constructor.
4. There are four ways of instantiating a date object:
 - a. `new Date()` //This will return the current date and time.
 - b. `new Date(milliseconds)`
 - c. `new Date(dateString)`
 - d. `new Date(year, month, day, hours, minutes, seconds, milliseconds)`
5. Here are a few examples of instantiating a date:
 - a. `var today = new Date()` //Current date
 - b. `var anotherDate = new Date("March 01, 2000 20:50:10")`
 - c. `var anotherDate = new Date(2000,03,01)`
 - d. `var anotherDate = new Date(2000,03,01,20,50,10)`

The following describes how to compare the two dates:

1. I have two dates. Date one is 15-01-2010 and another date is 15-01-2011.
2. I compare these two dates, using the following JavaScript code:

```
• <scripttype="text/javascript"language="javascript">
•
• function CompareDate() {
• //Note: 00 is month i.e. January
• var dateOne = new Date(2010, 00, 15); //Year, Month, Date
• var dateTwo = new Date(2011, 00, 15); //Year, Month, Date
• if (dateOne > dateTwo) {
• alert("Date One is greather then Date Two.");
• }else {
• alert("Date Two is greather then Date One.");
• }
• }
•
• CompareDate();
• </script>
```

Output



For more details, visit the following link:

- [How to Compare Two Dates Using JavaScript](#)

Question 35: What is Prototype Objects in JavaScript?

Answer - A Prototype Object is an object, which simplifies the process of adding the custom properties/methods to all the instances of an object. In JavaScript, you are allowed to add the custom properties to Prebuilt and Custom objects.

The following briefly describes Prebuilt and Custom Objects in JavaScript:

- *Prebuilt*: These are the objects, which are created with the new keyword such as an image, string, date, array object and so on.
- *Custom*: These are the objects, which are created by the developer to hold the properties or other information, for example:

Let's see some code in action to create and invoke the objects in JavaScript.

```

• <script language="javascript" type="text/javascript">
•     function Greet(mode) {
•         this.mode = mode;
•         this.callgreet = function() {
•             alert('Good ' + this.mode)
•         }
•     }
•     obj1 = new Greet("Morning")
•     obj1.callgreet() //alerts "Good Morning"
•     obj2 = new Greet("Evening")
•     obj2.callgreet() //alerts "Good Evening"
• </script>

```

For more details, visit the following link:

- [Prototype Objects in JavaScript](#)

Question 36: Explain Binding in JavaScript.

Answer:

Early binding

We can use a helper function, "bind", that forces this.

The following is an example of such a function. It accepts a function "func" and returns a wrapper, which calls "func" with this = "fixThis".

For example:

```

• function bind(func, fixThis) {
•     return function() {
•         return func.apply(fixThis, arguments)
•     }
• }

```

Late binding

Late binding is a variation of bind with a slightly different behavior.

In short, it means “binding on call time”, instead of “immediate binding”.

Late Binding in Action

To use late binding, we use "bindLate" instead of "bind".

For example:

```
• <!DOCTYPE HTML>
• <html>
•
• <body>
•   <script>
•       function bindLate(funcName, fixThis) { // instead
of bind
•           return function() {
•               return fixThis[funcName].apply(fixThis,
arguments)
•           }
•       }
•
•       function Mymenu(elem) {
•           this.Hello = function() {
•               alert('Mymenu')
•           }
•           elem.onclick = bindLate('Hello', this)
•       }
•
•       function BigMenu(elem) {
•           Mymenu.apply(this, arguments)
•           this.Hello = function() {
•               alert('BigMenu')
•           }
•       }
•
•       new BigMenu(document.body)
•   </script> Click here. I'm a BigMenu! </body>
•
• </html>
```

For more details, visit the following link:

- [Binding in JavaScript](#)

Question 37: How can you create a class and object of that class in JavaScript?

Answer - JavaScript is a prototype-based programming style of object-oriented programming in which classes are not present.

Creating Class in JavaScript

The following syntax is used for declaring a class in JavaScript:

```
• function Emp() {  
•     alert('Emp instantiated');  
• }
```

Here, Emp can act as a class in JavaScript.

1. Creating Objects of Emp Class

Using the following syntax, we can create an object of the Emp class:

```
var Emp1 = new Emp();
```

As soon as we create an object of the Emp class, the constructor will be called.

Here, the above function Emp would be treated as a class in JavaScript.

```

<!DOCTYPE html> Demo screen from
devesh omar
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>

<script >

function Emp() {
    alert('Emp instantiated');
}

var Emp1 = new Emp();

</script>
</head>
<body>
<form id="form1" runat="server">
<div>

</div>
</form>
</body>
</html>

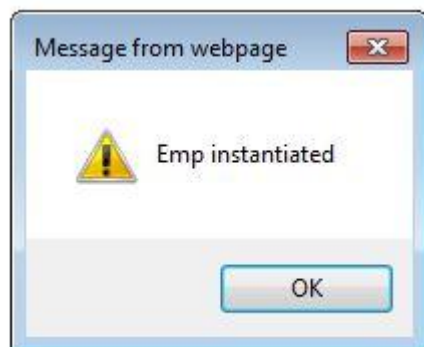
```

Emp act as an class

This function body act as Constructor of Emp

Creating objects of Emp class

2. Running the code



For more details, visit the following link:

- [Object Oriented Programming in JavaScript: Part 1](#)

Question 38: What is the callback approach in JavaScript?

Answer - The callback design approach is nothing but the implementation of a functional design approach. We know that there is no concrete class implementation

in JavaScript and everything is an object and obviously a function is also one type of an object in JavaScript. The USP of a function type of an object is "It is a first class object". In other words, it enjoys a higher priority in the Application.

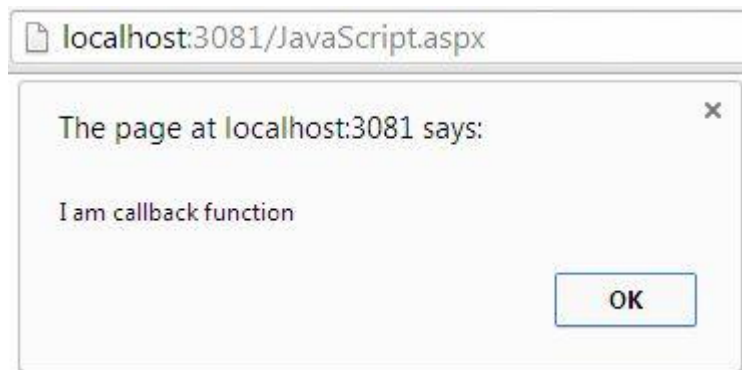
We can now pass a function as a parameter of another function. When the function (called) function finishes its execution, it can call another function (which is nothing but a callback). Hence, the function, that will execute after the execution of some other function is called the callback function. JavaScript and various JavaScript libraries (like jQuery) use callback functions every now and then.

Start with a sample example.

We will now implement a few callback functions in this section. Try to understand the following code:

```
• <%@ Page Language="C#" AutoEventWireup="true"
  CodeBehind="JavaScript.aspx.cs" Inherits="JavaScript.JavaS
  cript" %>
•
• <!DOCTYPE html>
• <html xmlns="http://www.w3.org/1999/xhtml">
•
• <head runat="server"> </head>
•
• <body>
•   <form id="form1" runat="server">
•     <script>
•       function call() {
•         alert("I am callback function");
•       }
•
•       function a(callBack) {
•         callBack();
•       }
•       //Passing function name as argument
•       a(call);
•     </script>
•   </form>
• </body>
•
• </html>
```

In this example, we have defined two functions ("call" and "a") and then we are calling a function by passing the name of the call() function. Thus, we are calling the callback function from function a. Here is the sample output:



For more details, visit the following link:

- [Advanced JavaScript: Callback Design Pattern and Callback Function](#)

Question 39: What is JavaScript Number Object?

Answer - Number object in JavaScript is used to represent the numeric values. Unlike other programming languages, JavaScript does not provide different types of numbers, like integer, float, short etc. In JavaScript, numbers are always stored as double precision floating point numbers. Numbers are stored in 64 bit format, where the fraction is stored in bits 0 to 51, the exponent in bits 52 to 62 and sign in bit 63. The number object is created as follows:

```
var value = new Number(numberValue);
```

Number Properties

The following are the properties of a Number.

MAX_VALUE	Returns largest possible value in JavaScript, 1.7976931348623157E+308
MIN_VALUE	Returns smallest possible value in JavaScript, 5E-324
NaN	Represents value that is not number

NEGATIVE_INFINITY	Represents value that is less than MIN_VALUE
POSITIVE_INFINITY	Represents value that is greater than MAX_VALUE

Number Methods

The following are the Number methods in JavaScript.

1. **toExponential():**

This method returns the string that represents a given number in the exponential format. It takes one optional parameter, an integer specifying the number of digits after the decimal points.

Syntax: x.toExponential(fractiondigits);

2. **toFixed():**

This method formats a number with the specific number of digits to the right of decimal point. It takes one optional parameter that represents the number of the digits to appear after the decimal points. If the parameter is not given, it is treated as 0.

Syntax: x.toFixed(digits);

3. **toPrecision():**

This method returns a string, with a number written with a specified length. It takes one optional parameter, which defines the number of the digits (including digits to the left and right of decimal point) to be displayed.

Syntax: x.toPrecision(digits);

4. **toString():**

This method returns the string representation of a number. It takes one optional parameter specifying the base for representing the numeric value. The parameter value should be between 2 to 36. If the parameter is not specified, the base is assumed to be 10.

Syntax: x.toString(radix);

5. **valueOf():**

This method returns the value of a number.

Syntax: x.valueOf();

For more details, visit the following link:

- [JavaScript Number Object](#)

Question 40: Explain try catch statement in JavaScript.

Answer - The try-catch Statement: In JavaScript, you need to write the code, where you can generate an error in the try block. Immediately after the try block, there is a catch block, which specifies the exception type that you want to catch. The syntax of the try-catch statement is as follows:

```
• Try {  
•     // code that can cause an error  
• }  
• Catch(err) {  
•     // What to do when an error occurs  
• }
```

In the preceding syntax, there are two blocks: try and catch. The try block contains the suspected code, that can generate the errors. If the error occurs, the try block throws an exception, which is caught by each block. There is another optional block known as a finally block, which is associated with the try-catch statement. The syntax of the finally block is as follows.

```
• Try {  
•     // code that can use an error  
• } catch (err) {  
•     // what to do when an error occurs  
• }  
• Finally  
• }  
• // code that executes in all cases  
• }
```

The optional finally block of the try-catch statement always run its code, whether or not exceptions are thrown. If the code in the try block runs completely, the finally block executes. If there is an error, the catch block executes and then the finally block executes. Any time a control is about to return to the caller from inside a try or catch block through an explicit return statement, the finally block executes just before the control returns.

For more details, visit the following link:

- [Handling Exceptions in JavaScript](#)

Question 41: What is the difference between Java and JavaScript?

Answer - Java and JavaScript are both Object Oriented Programming languages, but JavaScript is an object oriented scripting language. They are completely different programming languages with a small number of similarities. JavaScript is mainly used in Web pages. Almost all Java expression syntax and naming conventions are followed by JavaScript. This is the main reason for calling it JavaScript. Unlike Java, JavaScript does not bother about the methods, variables and classes at all. Java is much more complex than JavaScript. Let us compare Java and JavaScript in detail.

- In Java, the source code is first converted into an intermediate code, known as the byte code. This byte code is non-readable by humans and is independent of any machine. Afterwards, this byte code is interpreted by the interpreter (Java Virtual Machine). Since JavaScript is a scripting language, it is only interpreted. This is why making the changes in a Java program are more complex than making the changes in a JavaScript program.
- Java needs the Java Development Kit. JavaScript mainly requires a text editor.
- Java is a strongly typed language. On the other hand, JavaScript is a loosely typed language.
- In a Web environment, JavaScript is a front-end language, whereas Java is a back end language.
- JavaScript is considered as a part of HTML file, so it is visible in the source file. Java applets are not a part of HTML file so they are not visible in the source file.
- Java is very complex to learn due to its rules and restrictions whereas JavaScript is very easy to learn, as compared to Java.
- The User Interface of JavaScript is developed in HTML and is very easy to understand, whereas the User Interface of Java is developed in AWT or Swing, which is very complex to understand.
- The client-side is more secured in Java compared to JavaScript.
- In Java and JavaScript, routines are known as the methods and functions respectively.
- Java supports Polymorphism, but JavaScript does not support Polymorphism.
- Java uses the classes and objects to make its code reusable easily, but this service is not available in JavaScript.

For more details, visit the following link:

- [Java vs JavaScript](#)

Question 42: What are the advantages of JavaScript?

Answer - Advantages of JavaScript are-

1. Choose short and meaningful but readable variable names.
2. Avoid global variables, whenever possible.
3. Use one global object to encapsulate any global variables, you really need.
4. Always use var to declare your variables.
5. If you are using any loop like for each or while with the multiple statements, don't forget to use starting curly braces and ending curly braces. If you are writing with single statement, avoid starting and ending the curly braces.
6. Try to use the comments, if necessary, which will help you/others to understand better in later days.
7. Indent your code, so it's readable.
8. Use curly braces to define the blocks of the code.
9. Comment your code.
10. Always end your statement with a semi colon.
11. Always write the statements in one line because if you split the statement in multiple lines, for each line, the compiler places a semicolon at the end of a line, which is wrong. Thus, you have to be more careful, while writing.
12. If possible, try to use Typescript/ JavaScript++ rather JavaScript, because it checks syntaxes at the compile time and it supports OOPS features.
13. Maintains two kinds of JavaScript files, which are:
 - a. Minimized file for production
 - b. JavaScript file for development.
14. Be aware of an automatic semi-colon insertion, that happens.
15. Use for each rather than for loop.
16. Maintain JavaScript files light weight. This can be possible for each functionality. Use different JavaScript file.
17. Declare the variables outside loops.
18. Reduce DOM operations.
19. Be aware of the duplicate members (member field, member function).
20. Always place the script files reference at the end of the body tag.
21. Always separate JavaScript code from HTML file.
22. Always use === comparison.
23. Always try to avoid Eval() because it decreases the performance.

24. Always try to use {} instead of Object() because Object() decreases the performance.
25. If possible, use short hand notations.
26. Use Switch case, instead of too many, if else blocks.
27. Use anonymous functions, instead of naming the functions, if possible.
28. Cache your selectors in the variables for further use.
29. Do not mix CSS properties with JavaScript / JQuery. Instead, we can use CSS classes in JavaScript / JQuery.
30. Before writing any condition with DOM element, first check, whether the DOM element is defined or not.

For more details, visit the following link:

- [Best Practices: JavaScript, CSS and HTML](#)

Question 43: Explain the concept of chaining in JavaScript.

Answer - This technique got popularity from jQuery. We write the series of statements, one after the other like a chain.

ex,

- `$("#h1").text("Change text").css("color", "red");`
- Another example of chaining **while** working **with** strings:
- `var s="hello";`
- `s.substring(0,4).replace('h','H'); //Hell`

The Chaining Method is also known as Cascading, because it repeatedly calls one method on an object, forming a chain/continuous line of the code.

Implementation of Chaining:

When a method returns this, the entire object is returned & it is passed to the next method and it is called chaining. Example,

- `var games = function() {`
- `this.name = '';`
- `}`
- `games.prototype.getName = function() {`
- `this.name = 'Age of Empire';`
- `return this;`
- `}`

```

• games.prototype.setName = function(n) {
•     this.name = n;
• }
• //now execute it
• var g = new games();
• g.getName().setName('Angry Birds');
• console.log(g.name);

```

We've created getName method, which returns this and implemented chaining.

Advantages

- Code is more maintainable, simple, lean.
- It is easy to read chaining code.

For more details, visit the following link:

- [Voice of a Developer: JavaScript Chaining - Part Sixteen](#)

Question 44: What are the Pure and Impure JavaScript functions?

Answer - Pure function

These are the functions, which always return the same value, when given the same arguments. They take some parameters, return a value, based on these, but don't change the parameter values. Example product is a function, which will always give the same output, depending upon the input.

```

• function product(a, b) {
•     return a * b;
• }
• console.log(product(2, 3)); // always give 6 whenever you
    pass 2, 3

```

Another nice property of the pure function is that, it doesn't modify the states of the variables out of its scope.

Impure function

Impure function may have the side effects and may modify the arguments, which are passed to them. The return value will depend upon the arguments. There may be a case, where for the same arguments, you'll get different values:

Example

```

• var count = 0;
•
• function Hits() {
•     count += 1;
• }
• Hits(); // will make count 1
• Hits(); // will make count 2
• Hits(); // will make count 3

```

Here, it's using an external variable count and also modifying it. If a function has the side effects, whether it updates any file or a database, it also falls under the category of an impure function.

Summary of differences

No side effects like update or db calls	May have side effects
Don't modify arguments which are passed to them	May modify arguments passed to them,
Always return the same value	Even if you call with same arguments, you may get different values.

For more details, visit the following link:

- [Voice of a Developer: JavaScript Pure And Impure Function - Part Thirteen](#)

Question 45: Explain “this” and “delete” keyword in JavaScript.

Answer - Delete keyword

When we work on the large Applications, we work with many objects. We are familiar with the problems like memory leak, performance of an Application is not good and similar issues arise at a later stage. I always recommend, that we need to know, how much memory, are we consuming in our Application.

Ex- Chrome provides a quick way to look into the memory usage.

The main advantage is to delete an unwanted property from an object.

- `var automobile = {engine: true, fuel: true, tyres:2 };`
- `var car = Object.create (automobile); //car is instanceof automobile`
- `console.log (car.tyres); // 2 it inherits from parent class`
- `//as we know car has own tyres then we can create own property of car object`
- `car.tyres = 4; // will make own tyre property of car object`

This keyword

“this” is not a variable. It is a keyword. You cannot change the value of this. It refers to the global object in all the global code.

We know, how to create the multiple objects. We can use Object Constructor function. We'll use this keyword to create a property, which belongs to a function.

- `function Automobile(engine, fuel, tyres) {`
- `this.engine = engine;`
- `this.fuel = fuel;`
- `this.tyres = tyres;`
- `}`
- `var car = new Automobile(true, true, 4);`
- `var bike = new Automobile(true, true, 2);`

This refers to the parent object in the body of the function:

- `var counter = {`
- `val: 0,`
- `increment: function() {`
- `this.val += 1;`
- `}`
- `};`
- `counter.increment();`
- `console.log(counter.val); // 1`
- `counter['increment']();`
- `console.log(counter.val); // 2`

For more details, visit the following link:

- [Voice of a Developer: Part Nine - JavaScript Useful Reserved Keywords](#)

Question 46: Is JavaScript an object oriented language?

Answer - Yes it is, but many developers differ because there is no “class” keyword in it. I believe JavaScript has all the ingredients to be an OOP language. We can read in detail OOP in JavaScript

Encapsulation

Encapsulate means “enclose (something) in or as if in a capsule”. It refers to enclosing all the functionalities within that object, so it is hidden from the rest of the Application. There are many ways to create the Objects (refer <http://www.c-sharpcorner.com/article/voice-of-a-developer-javascript-objects/> Voice of a Developer – part 2 for the details).

Below is the ubiquitous object literal pattern, my favorite way to declare the objects:

```
• var myObj = {name: "Sumit", profession: "Developer",  
  getDetails: function(n, p){  
•   return n+ ' ' +p;  
•   }  
• };
```

Inheritance

JavaScript supports prototypal inheritance rather than classical inheritance. We studied this in part 2. Let us dig into it in further detail. There are two concepts:

Prototype property:

Every JavaScript function has a prototype property and you attach the properties and methods on this prototype property, when you want to implement inheritance. Firefox and most versions of Safari and Chrome have a `__proto__` “pseudo”, which we have learned in part 2. There is another property called prototype property for an inheritance.

Prototype attribute:

JS runtime first looks property on the object. If not found, looks for the property on the object’s prototype—the object it inherited, its properties from. This is called as prototype chaining. JavaScript uses this prototype chain to look for the properties and methods of an object.

Abstraction

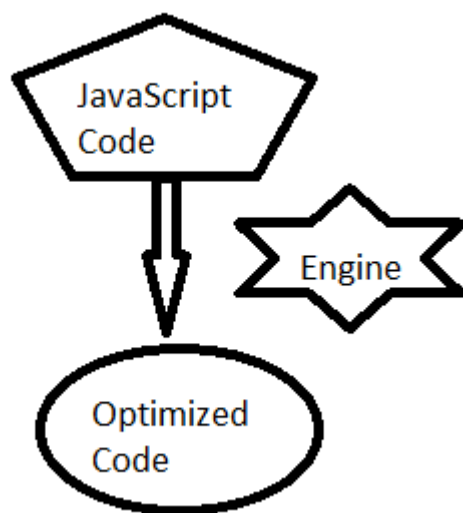
It means to hide certain details and show the required objects. This improves the readability and maintainability of the code. For a good code quality and to reduce risk of an object getting modified outside accidentally, it is good to adopt an abstraction.

For more details, visit the following link:

- [Voice of a Developer: Part Eight - JavaScript OOP](#)

Question 47: What is a JavaScript engine? How JavaScript engine works?

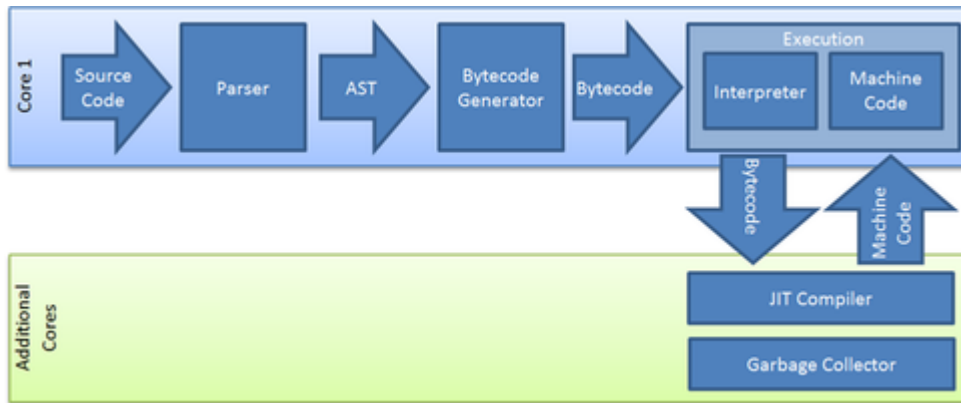
Answer - It's a program or a library, which executes JavaScript code.



Evolution of JavaScript engines

The 1st JavaScript engine was created by Brendan Eich in late 1990s for Netscape. It was named SpiderMonkey and was written in C++.

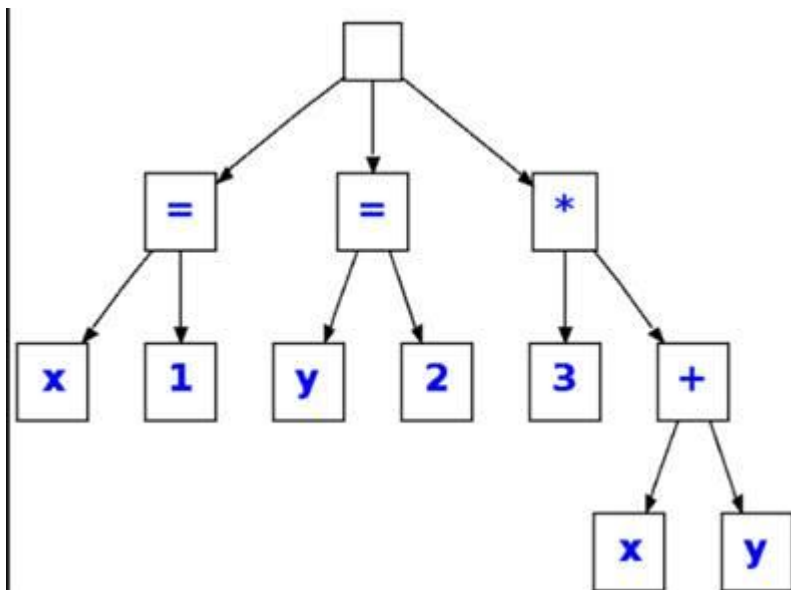
Here is the process of working:



The parser takes Javascript and generates an Abstract Syntax Tree (AST).

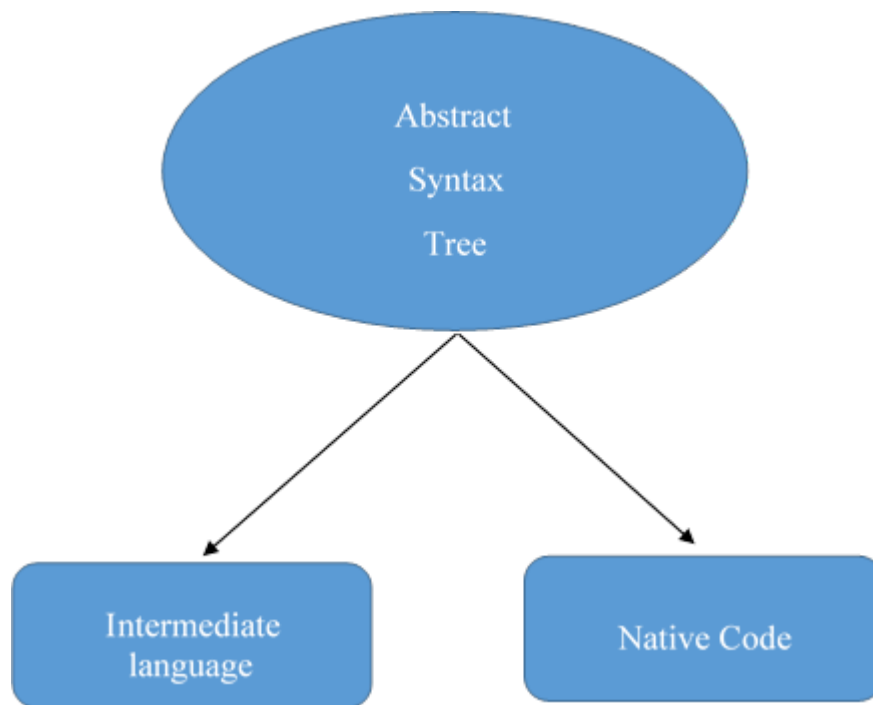
AST: It's just a syntax tree and a tree representation of the source code:

ex:



After AST is generated, there are two choices, depending upon JavaScript engine to generate the bytecode.

For example -- Google V8 & Chakra generates Native code and both are really fast engines to run JavaScript.



For more details, visit the following link:

- [Voice of a Developer: Javascript Engines - Part Three](#)

Question 47: What mistakes do developers generally make in JavaScript code?

Answer - As a developer, sometimes, I face issues because of my silly programming mistakes and I have learned from these. There can be the following mistakes-

Equality checks

We check the variables for certain values, but there are two types of operators, we can use. I suggest using strict operators over type converting.

Type converting operator like `==` converts operands, if they are not of same type. To demonstrate or test, you can try the following statements and see the output.

- `== 1 // true`
- `"1" == 1 // true`
- `== '1' // true`
- `== false // true`

So, the mistake of using == operator results in TRUE value for 1=="1", which can be wrong in a real scenario. Therefore, it is always advisable to use type strict operators.

Strict operators like === don't convert operands and returns true, if the operands are strictly equal.

- === 1 // true
- "1" === 1 // false
- === false // true

Concatenation

Javascript uses + operator for concatenation & addition. Now, another common mistake is to use mix numbers & string operands. For example-

- var y = '10';
- var x= 1 + y; // will give 110

Use function parseInt in such scenario to rescue you, otherwise you'll spend time in debugging.

- var x= 1 + parseInt(y); // will give 11

Float point numbers

Programming languages are not always correct to match the floating point numbers. Let's see right way to match the floating point numbers. For example-

- var f = 0.1;
- var g = 0.2;
- var h= f + g;
- (h === 0.3) // false because h is 0.30000000000000004

Floating numbers are saved in 64 bits. Hence, the right way to do is:

- var h = (f * 10 + g * 10) / 10; // h is 0/3
- (h === 0.3) // true

Usage of undefined as a variable

Often, we use undefined as a variable. Let's take a scenario, given below:

```
> myVar === undefined
```

```
✖ ▶ Uncaught ReferenceError: myVar is not defined(...)
```

The correct way is to use typeof, when checking for undefined:

```
> typeof myVar === "undefined"
< true
```

For more details, visit the following link:

- [Voice of a Developer: Common Mistakes - Part Four](#)

Question 48: How many data types in JavaScript?

Answer - There are two types of data types in JavaScript.

Primitive data type

- String
- Number
- Boolean
- Undefined
- Null

Non-primitive (reference) data type

- Object
- Array
- RegExp

For more details, visit the following link:

- [Voice of a Developer: JavaScript Data Types](#)

Question 49: How can you declare a function in JavaScript and its hoisting?

Answer - The traditional way of defining a function is:

```
• function logMe() //function name logMe
• {
•     console.log('Log me in');
• }
• logMe();
```

If you skip the function name, it'll become an anonymous function. We'll talk about them in detail later but here is the code:

```
• (function() {  
•     console.log('Log me in');  
• })(); // we're calling this anonymous function;
```

Function hoisting

It is the process in which JavaScript runtime hoists all the functions declared, using the function declaration syntax at the top of JavaScript file. Look at the example, given below:

```
• function logMe() {  
•     console.log('Log me in');  
• }  
• logMe();  
•  
• function logMe() {  
•     console.log('Log me again');  
• }  
• logMe();
```

FUNCTION EXPRESSION

It's the second form of declaring a function.

```
• var log = function() {  
•     console.log('Log me in');  
• }
```

Now, writing logMe after the function is futile, because it can now be called, using the assigned variable log.

Consider the following example:

```
• var log = function logMe() {  
•     console.log('Log me in');  
• }  
• log();  
• var log = function logMe() {  
•     console.log('Log me again');  
• }  
• log();
```

For more details, visit the following link:

- [Voice of a Developer: JavaScript Functions Invocations - Part 11](#)

Question 50: What are the Optional Parameters in JavaScript?

Answer - Optional Parameters are a great feature in a programming language. It helps the programmers to pass a fewer number of parameters to a function and assign the default value. For instance, there is a function “getEmployee(var id, var name)” which takes two params ID and name. However, we will pass only one param as ID. Now, we try to access the name variable. Inside a function, it throws us an exception, which is undefined. In order to avoid the problem, we need to implement Optional Parameters. In JavaScript, it can be implemented in the following ways:

1. Using undefined property
2. Using arguments variable
3. Using OR(||) operator

Using undefined property

Undefined property indicates that a value is not assigned to a variable. By using undefined, we can handle Optional Parameters in JavaScript.

Let's analyze the following example:

```
• function browseBlog(blogURL, type) {
•     if (blogURL === undefined) {
•         blogURL = "DefaultURL";
•     }
•     if (type === undefined) {
•         type = "DefaultType";
•     }
•     alert(blogURL);
•     alert(blogType);
• }
• browseBlog("www.c-sharpcorner.com", "Microsoft"); // two
  alerts with "www.c-sharpcorner.com", "Microsoft"
• browseBlog("www.c-sharpcorner.com"); // two alerts with
  "www.c-sharpcorner.com", "DefaultType"
```

Using arguments variable

JavaScript functions has a built-in object called arguments. It is nothing, but contains an array of the parameters, when function is invoked. You can loop over the arguments object and find out all the parameters. Let's see:

```
• function browseBlog(blogURL, type) {  
•   if (arguments.length == 0) // Means no parameters are  
    passed  
•   {  
•       blogURL = "DefaultURL";  
•       Type = "DefaultBlog";  
•   }  
•   if (arguments.length == 1) // Means second parameter  
    is not passed  
•   {  
•       Type = "DefaultType";  
•   }  
•   alert(blogURL);  
•   alert(blogType);  
•   // Get all parameters  
•   for (i = 0; i < arguments.length; i++) {  
•       alert(arguments[i]);  
•   }  
• }  
• browseBlog("www.c-sharpcorner.com", "Microsoft"); //  
    alerts two times with value "www.c-sharpcorner.com",  
    "Microsoft"  
• browseBlog("www.c-sharpcorner.com"); // alerts two times  
    with value "www.c-sharpcorner.com", "DefaultType"
```

Using OR (||) operator

The short-circuit OR operator || returns the left side, if the left argument is true (evaluates to true in conditionals), else it checks, if the right argument is true and returns it. We can use this shortcut operator to manage the optional parameters in JavaScript. This method only allows the last arguments to be optional and you cannot make an optional first parameter, middle parameters are optional.

Let's analyze the following example:

```
• function browseBlog(blogURL, type) {  
•   alert(blogURL);
```

```
•   var blogType = type || "Default";
•   alert(blogType);
• }
• browseBlog("www.c-sharpcorner.com", "Microsoft"); //
  alerts two times with value "www.c-sharpcorner.com",
  "Microsoft"
• browseBlog("www.c-sharpcorner.com"); // alerts two times
  with value "www.c-sharpcorner.com", "Default"// This is
  just a sample script. Paste your real code (javascript or
  HTML) here.
• if ('this_is' == /an_example/) {
•   of_beautifier();
• } else {
•   var a = b ? (c % d) : e[f];
• }
```