JAVASCRIPT INTERVIEW QUESTIONS FOR BEGINNERS TO EXPERIENCED

ECMA-5,6,7,8

1) What is JavaScript?

JavaScript is *a scripting language*. It is different from Java language. It is object-based, lightweight, cross-platform translated language. It is widely used for client-side validation. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

2) List some features of JavaScript?

Some of the features of JavaScript are:

- JavaScript is a lightweight, interpreted programming language.
- JavaScript is designed for creating network-centric applications.
- JavaScript is complementary to and integrated with Java.

- JavaScript is is complementary to and integrated with HTML.
- JavaScript is open and cross-platform.

3) List some of the advantages of JavaScript?

Some of the advantages of JavaScript are:

- **Less server interaction :** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors :** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity :** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces :** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

4) List some of the disadvantages of JavaScript?

Some of the disadvantages of JavaScript are:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript can not be used for Networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocess capabilities.

5) Name the types of functions?

The types of function are:

- **Named :** These type of functions contains name at the time of definition.
- **Anonymous :** These type of functions doesn't contain any name. They are declared dynamically at runtime.

6) Define a named function in JavaScript?

The function which has named at the time of definition is called a named function. For example:

```
function msg()
{
  document.writeln("Named Function");
}
msg();
```

7) Define anonymous function in JavaScript?

It is a function that has no name. These functions are declared dynamically at runtime using the function operator instead of the function declaration. The function operator is more flexible than a function declaration. It can be easily used in the place of an expression. For example:

```
var display=function()
{
  alert("Anonymous Function is invoked");
}
display();
```

8) Can an anonymous function be assigned to a variable?

Yes, you can assign an anonymous function to a variable.

9) In JavaScript what is an argument object?

The variables of JavaScript represent the arguments that are passed to a function.

10) Define closure?

In JavaScript, we need closures when a variable which is defined outside the scope in reference is accessed from some inner scope.

```
function create() {
   var counter = 0;
   return {
      increment: function() {
         counter++;
      },

      print: function() {
         console.log(counter);
      }
   }
}
var c = create();
c.increment();
c.print();     // ==> 1
```

11) If we want to return the character from a specific index which method is used?

The JavaScript string charAt() method is used to find out a char value present at the specified index. The index number starts from 0 and goes to n-1, where n is the length of the string. The index value can't be a negative, greater than or equal to the length of the string. For example:

```
var str="Javatpoint";
document.writeln(str.charAt(4));
```

12) What is the difference between JavaScript and JScript?

Netscape provided the JavaScript language. Microsoft changed the name and called it JScript to avoid the trademark issue. In other words, you can say JScript is the same as JavaScript, but Microsoft provides it.

13) How to write a hello world example of JavaScript?

A simple example of JavaScript hello world is given below. You need to place it inside the body tag of HTML.

```
<script type="text/javascript">
document.write("JavaScript Hello World!");
</script>
```

14) How to use external JavaScript file?

I am assuming that js file name is message.js, place the following script tag inside the head tag.

```
<script type="text/javascript" src="message.js"></script>
```

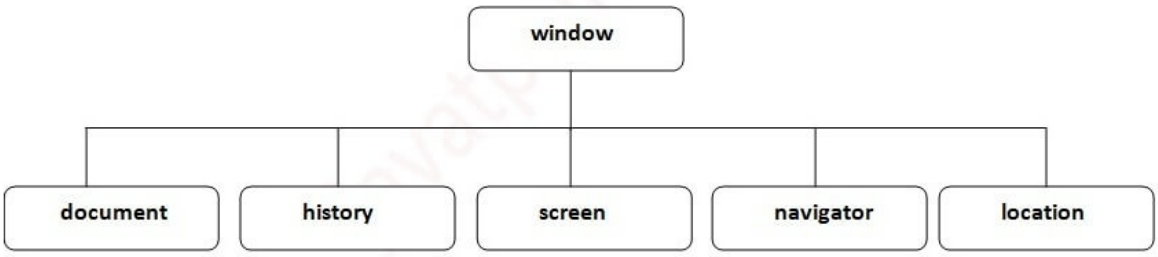
15) Is JavaScript case sensitive language?

Yes, JavaScript is a case sensitive language. For example:

```
Var msg = "JavaScript is a case-sensitive language"; //Here, var should be used to declare a variable
function display()
{
document.writeln(msg); // It will not display the result.
}
display();
```

16) What is BOM?

BOM stands for *Browser Object Model*. It provides interaction with the browser. The default object of a browser is a window. So, you can call all the functions of the window by specifying the window or directly. The window object provides various properties like document, history, screen, navigator, location, innerHeight, innerWidth.



17) What is the use of window object?

The window object is created automatically by the browser that represents a window of a browser. It is not an object of JavaScript. It is a browser object. The window object is used to display the popup dialog box. Let's see with description:

- **open() :** opens the new window.

```
<script type="text/javascript">
function msg(){
open("http://www.javatpoint.com");
}
</script>
<input type="button" value="javatpoint" onclick="msg()"/>
```

- **close() :** closes the current window.

- **setTimeout() :** performs action after specified time like calling function, evaluating

expressions etc.

```
<script type="text/javascript">
function msg(){
setTimeout(
function(){
alert("Welcome to Javatpoint after 2 seconds")
},2000);

}
</script>

<input type="button" value="click" onclick="msg()"/>
```

18) What is the use of history object?

The history object of a browser can be used to switch to history pages such as back and forward from the current page or another page. The history object is the window property, so it can be accessed by:

```
window.history
```

Or,

```
history
```

There are three methods of history object are given below:

- **back() :** It loads the previous page.

```
<html>
<head>
<script>
function goBack() {
  window.history.back()
}
</script>
</head>
<body>

<input type="button" value="Back" onclick="goBack()">

</body>
</html>
```

- **forward() :** It loads the next page.

```
<html>
<head>
<script>
function goForward() {
  window.history.forward()
}
</script>
</head>
<body>

<input type="button" value="Forward" onclick="goForward()">

</body>
</html>
```

- **go(number) :** The number may be positive for forward, negative for backward. It

loads the given page number.

19) What is the use of navigator object?

The JavaScript navigator object is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc. The navigator object is the window property, so it can be accessed by:

```
window.navigator
```

Or,

```
navigator
```

There are many properties of navigator object that returns information of the browser are given below:

- **cookieEnabled :** returns true if cookie is enabled otherwise false.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"cookiesEnabled is " + navigator.cookieEnabled;
</script>
```

- **appName :** returns the name.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"navigator.appName is " + navigator.appName;
</script>
```

- **appCodeName :** returns the code name.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"navigator.appCodeName is " + navigator.appCodeName;
</script>
```

- **appVersion :** returns the version.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = navigator.appVersion;
</script>
```

- **userAgent :** returns the user agent.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = navigator.userAgent;
</script>
```

- **platform :** returns the platform e.g. Win32.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = navigator.platform;
</script>
```

- **language :** returns the language. It is supported in Netscape and Firefox only.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = navigator.language;
</script>
```

- **onLine :** returns true if browser is online otherwise false.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = navigator.onLine;
</script>
```

- **javaEnabled() :** checks if java is enabled.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = navigator.javaEnabled();
</script>
```

20) What is the use of location object?

The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page. The window.location object can be written without the window prefix.

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

Display the href (URL) of the current page:

```
document.getElementById("demo").innerHTML =
"Page location is " + window.location.href;
```

Result is:

```
Page location is https://www.w3schools.com/js/js_window_location.asp
```

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

Display the name of the host:

```
document.getElementById("demo").innerHTML =
"Page hostname is " + window.location.hostname;
```

Result is:

```
Page hostname is www.w3schools.com
```

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

Display the path name of the current URL:

```
document.getElementById("demo").innerHTML =
"Page path is " + window.location.pathname;
```

Result is:

```
Page path is /js/js_window_location.asp
```

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

Display the web protocol:

```
document.getElementById("demo").innerHTML =
"Page protocol is " + window.location.protocol;
```

Result is:

```
Page protocol is https:
```

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

```
<html>
<head>
<script>
function newDoc() {
  window.location.assign("https://www.w3schools.com")
}
</script>
</head>
<body>

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

</body>
</html>
```

21) What is the use of screen object?

- The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc. The navigator object is the window property, so it can be accessed by:

```
window.screen
```

Or,

```
screen
```

- There are many properties of screen object that returns information of the browser.

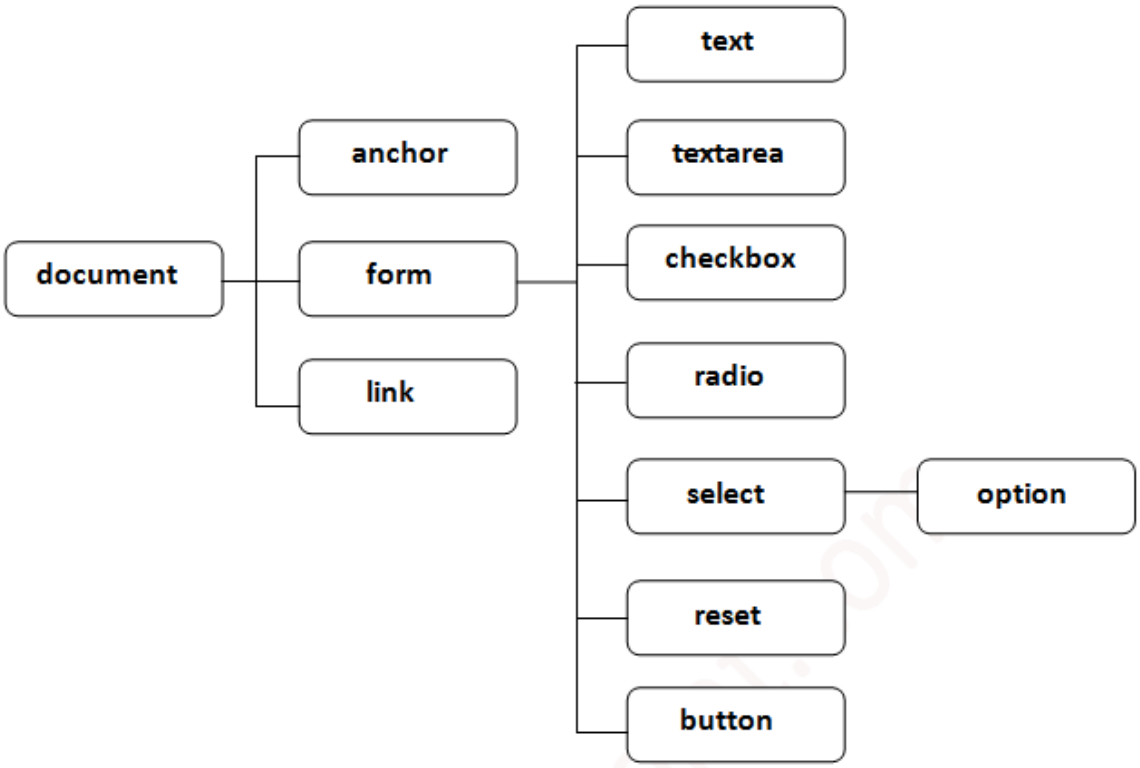
1	width	returns the width of the screen
2	height	returns the height of the screen
3	availWidth	returns the available width
4	availHeight	returns the available height
5	colorDepth	returns the color depth
6	pixelDepth	returns the pixel depth.

```
<script>
document.writeln("<br/>screen.width: "+screen.width);
document.writeln("<br/>screen.height: "+screen.height);
document.writeln("<br/>screen.availWidth: "+screen.availWidth);
document.writeln("<br/>screen.availHeight: "+screen.availHeight);
document.writeln("<br/>screen.colorDepth: "+screen.colorDepth);
document.writeln("<br/>screen.pixelDepth: "+screen.pixelDepth);
</script>
```

22) What is DOM? What is the use of document object?

DOM stands for *Document Object Model*. A document object represents the HTML

document. It can be used to access and change the content of HTML. In other words the HTML DOM is a standard for how to get, change, add, or delete HTML elements. Let's see the properties of document object that can be accessed and modified by the document object.



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page.
- JavaScript can change all the HTML attributes in the page.
- JavaScript can change all the CSS styles in the page.
- JavaScript can remove existing HTML elements and attributes.
- JavaScript can add new HTML elements and attributes.
- JavaScript can react to all existing HTML events in the page.

- JavaScript can create new HTML events in the page.

23) Accessing field value by document object?

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

- Here, **document** is the root element that represents the html document.
- form1 is the name of the form.
- name is the attribute name of the input text.
- value is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>

<form name="form1">
Enter Name:<input type="text" name="name"/>
<input type="button" onclick="printvalue()" value="print name"/>
</form>
```

24) Finding HTML elements by document object?

- document.getElementById(*id*) : Find an element by element id.

```
<script type="text/javascript">
function getcube(){
var number=document.getElementById("number").value;
alert(number*number*number);
}
</script>
<form>
Enter No:<input type="text" id="number" name="number"/><br/>
<input type="button" value="cube" onclick="getcube()"/>
</form>
```

- document.getElementsByTagName(*name*) : Find elements by tag name.

```
<script type="text/javascript">
function countpara(){
var totalpara=document.getElementsByTagName("p");
alert("total p tags are: "+totalpara.length);

}
</script>
```

```
<p>This is a pragraph</p>
<p>Here we are going to count total number of paragraphs by getElementByTagName() method.</p>
<p>Let's see the simple example</p>
<button onclick="countpara()">count paragraph</button>
```

- **document.getElementsByClassName(*name*) :** Find elements by class name.

```
<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the <b>getElementsByClassName</b> method.
</p>

<p id="demo"></p>

<script>
var x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;
</script>
```

- document.getElementsByName(*name*) : Find elements by name.

```
<script type="text/javascript">
function totalelements()
{
var allgenders=document.getElementsByName("gender");
alert("Total Genders:"+allgenders.length);
}
</script>
<form>
```

```
Male:<input type="radio" name="gender" value="male">
Female:<input type="radio" name="gender" value="female">

<input type="button" onclick="totalelements()" value="Total Genders">
</form>
```

- **querySelectorAll():** find all HTML elements that match a specified CSS selector.

```
<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the <b>querySelectorAll</b> method.</p>

<p id="demo"></p>

<script>
var x = document.querySelectorAll("p.intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;
</script>
```

25) Changing HTML elements by document object?

- **element.innerHTML = new html content :** Change the inner HTML of an element. use this syntax [document.getElementById(id).innerHTML = new HTML].

```
<script type="text/javascript" >
function showcommentform() {
var data="Name:<input type='text' name='name'><br>Comment:<br><textarea rows='5' cols='80'>
</textarea>
<br><input type='submit' value='Post Comment'>";
document.getElementById('mylocation').innerHTML=data;
}
```

```
</script>
<form name="myForm">
<input type="button" value="comment" onclick="showcommentform()">
<div id="mylocation"></div>
</form>
```

- **element.attribute = new value :** Change the attribute value of an HTML element. use this syntax [document.getElementById(id).attribute = new value].

```
<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>
```

- **element.style.property = new style :** Change the style of an HTML element. use this syntax [document.getElementById(id).style.property = new style].

```
<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>
```

26) Adding and Deleting HTML elements by document object?

- **document.createElement(*element*) :** Create an HTML element.
- **document.appendChild(*element*) :** Add an HTML element.

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
```

```
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

- **document.removeChild(*element*) :** Remove an HTML element.

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>
```

```
<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

- **document.replaceChild(*new, old*) :** Replace an HTML element.

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>

var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```

27) Adding and Deleting event handlers by document object?

- **addEventListener():** method attaches an event handler to the specified element.

```
<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
document.getElementById("myBtn").addEventListener("click", displayDate);

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
```

- **removeEventListener():** method removes event handlers that have been attached with the addEventListener() method.

```
<div id="myDIV">
  <p>This div element has an onmousemove event handler that displays a random number
every time you move your mouse inside this orange field.</p>
  <p>Click the button to remove the div's event handler.</p>
  <button onclick="removeHandler()" id="myBtn">Remove</button>
</div>

<p id="demo"></p>
```

```
<script>
document.getElementById("myDIV").addEventListener("mousemove", myFunction);

function myFunction() {
  document.getElementById("demo").innerHTML = Math.random();
}

function removeHandler() {
  document.getElementById("myDIV").removeEventListener("mousemove", myFunction);
}
</script>
```

28) What is Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM, bubbling and capturing. Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a element, and the user clicks on the <p> element, which element's "click" event should be handled first?

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

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

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter **addEventListener(*event*, *function*, *useCapture*);** The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

```
<div id="myDiv1">
  <h2>Bubbling:</h2>
  <p id="myP1">Click me!</p>
</div><br>

<div id="myDiv2">
  <h2>Capturing:</h2>
  <p id="myP2">Click me!</p>
</div>
```

```
<script>
document.getElementById("myP1").addEventListener("click", function() {
  alert("You clicked the white element!");
}, false);

document.getElementById("myDiv1").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, false);

document.getElementById("myP2").addEventListener("click", function() {
  alert("You clicked the white element!");
}, true);

document.getElementById("myDiv2").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, true);
</script>
```

29) How to write a comment in JavaScript?

There are two types of comments in JavaScript.

- Single Line Comment: It is represented by // (double forward slash)
- Multi-Line Comment: Slash represents it with asterisk symbol as /* write comment here */

30) How to create a function in JavaScript?

To create a function in JavaScript, follow the following syntax:

```
function function_name(){
//function body
}
```

31) What are the JavaScript data types?

There are two types of data types in JavaScript:

- **Primitive Data Types :** The primitive data types are as follows:

Data Type	Description
String	represents a sequence of characters, e.g., "hello"
Number	represents numeric values, e.g., 100
Boolean	represents boolean value either false or true
Undefined	represents an undefined value
Null	represents null, i.e., no value at all

- **Non-primitive Data Types :** The non-primitive data types are as follows:

Data Type	Description
Object	represents an instance through which we can access members
Array	represents a group of similar values
RegExp	represents regular expression

32) What is the difference between == and ===?

The == operator checks equality only whereas === checks equality, and data type, i.e., a value must be of the same type.

33) How to write HTML code dynamically using JavaScript?

The innerHTML property is used to write the HTML code using JavaScript dynamically. Let's see a simple example:

- **document.getElementById('mylocation').innerHTML="<h2>This is heading using JavaScript</h2>";**

34) How to write normal text code using JavaScript dynamically?

The innerText property is used to write the simple text using JavaScript dynamically. Let's see a simple example:

- **document.getElementById('mylocation').innerText="This is text using JavaScript";**

35) How to create objects in JavaScript?

There are 3 ways to create an object in JavaScript.

- **By object literal :** object={property1:value1,property2:value2.....propertyN:valueN}

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

- **By creating an instance of Object :** var objectname=new Object();

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

- **By Object Constructor :** The this keyword refers to the current object.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

36) How to create an array in JavaScript?

There are 3 ways to create an array in JavaScript.

- **By array literal :** var arrayname=[value1,value2.....valueN];

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

- **By creating an instance of Array :** var arrayname=new Array();

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

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

- **By using an Array constructor :** create instance of array by passing arguments in constructor.

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

37) What does the isNaN() function?

The isNan() function returns true if the variable value is not a number. For example:

```
function number(num) {
  if (isNaN(num)) {
    return "Not a Number";
  }
  return "Number";
}
console.log(number('1000F'));
// expected output: "Not a Number"

console.log(number('1000'));
// expected output: "Number"
```

38) What is the output of 10+20+"30" in JavaScript?

3030 because 10+20 will be 30. If there is numeric value before and after +, it treats as binary + (arithmetic operator).

```
function display()
{
 document.writeln(10+20+"30");
}
display();
```

39) What is the output of "10"+20+30 in JavaScript?

102030 because after a string all the + will be treated as string concatenation operator (not binary +).

```
function display()
{
 document.writeln("10"+20+30);
}
display();
```

40) Difference between Client side JavaScript and Server side JavaScript?

- **Client-side JavaScript :** comprises the basic language and predefined objects which are relevant to running JavaScript in a browser. The client-side JavaScript is embedded directly by in the HTML pages. The browser interprets this script at runtime.

- **Server-side JavaScript :** also resembles client-side JavaScript. It has a relevant JavaScript which is to run in a server. The server-side JavaScript are deployed only after compilation.

41) In which location cookies are stored on the hard disk?

- The storage of cookies on the hard disk depends on the OS and the browser.
- The Netscape Navigator on Windows uses a cookies.txt file that contains all the cookies. The path is c:\Program Files\Netscape\Users\username\cookies.txt
- The Internet Explorer stores the cookies on a file username@website.txt. The path is: c:\Windows\Cookies\username@Website.txt.

42) What is the real name of JavaScript?

The original name was **Mocha**, a name chosen by Marc Andreessen, founder of Netscape. In September of 1995, the name was changed to LiveScript. In December 1995, after receiving a trademark license from Sun, the name JavaScript was adopted.

43) What is the difference between undefined value and null value?

- **Undefined value :** A value that is not defined and has no keyword is known as undefined value. For example:

```
int number;//Here, a number has an undefined value.
```

- **Null value :** A value that is explicitly specified by the keyword "null" is known as a null value. For example:

```
String str=null;//Here, str has a null value.
```

44) How to set the cursor to wait in JavaScript?

The cursor can be set to wait in JavaScript by using the property "cursor". The following example illustrates the usage:

```
<script>
window.document.body.style.cursor = "wait";
</script>
```

45) What is this [[[]]]?

This is a three-dimensional array.

```
var myArray = [[[]]];
```

46) Are Java and JavaScript same?

No, Java and JavaScript are the two different languages. Java is a robust, secured and object-oriented programming language whereas JavaScript is a client-side scripting language with some limitations.

47) What is negative infinity?

Negative Infinity is a number in JavaScript which can be derived by dividing the negative number by zero. For example:

```
var num=-5;
function display()
{
  document.writeln(num/0);
}
display();
//expected output: -Infinity
```

48) What is the difference between View state and Session state?

"View state" is specific to a page in a session whereas "Session state" is specific to a user or browser that can be accessed across all pages in the web application.

49) What are the pop-up boxes available in JavaScript?

- **Alert Box :** displays the alert box containing message with ok button.

```
<script type="text/javascript">
function msg(){
 alert("Hello Alert Box");
}
</script>
<input type="button" value="click" onclick="msg()"/>
```

- Confirm Box : displays the confirm dialog box containing message with ok and cancel button.

```
<script type="text/javascript">
function msg(){
var v= confirm("Are u sure?");
if(v==true){
alert("ok");
}
else{
alert("cancel");
```

```
}

}
</script>

<input type="button" value="delete record" onclick="msg()"/>
```

- **Prompt Box :** displays a dialog box to get input from the user.

```
<script type="text/javascript">
function msg(){
var v= prompt("Who are you?");
alert("I am "+v);

}
</script>

<input type="button" value="click" onclick="msg()"/>
```

50) How can we detect OS of the client machine using JavaScript?

The **navigator.appVersion** string can be used to detect the operating system on the client machine.

51) How to submit a form using JavaScript by clicking a link?

Let's see the JavaScript code to submit the form by clicking the link.

```
<form name="myform" action="index.php">
Search: <input type='text' name='query' />
<a href="javascript: submitform()">Search</a>
</form>
<script type="text/javascript">
```

```
function submitform()
{
  document.myform.submit();
}
</script>
```

52) Is JavaScript faster than ASP script?

Yes, because it doesn't require web server's support for execution.

53) How to change the background color of HTML document using JavaScript?

```
<script type="text/javascript">
document.body.bgColor="pink";
</script>
```

54) How to handle exceptions in JavaScript?

By the help of try/catch block, we can handle exceptions in JavaScript. JavaScript supports try, catch, finally and throw keywords for exception handling.

55) How to validate a form in JavaScript?

```
<script>
function validateform(){
var name=document.myform.name.value;
var password=document.myform.password.value;

if (name==null || name==""){
  alert("Name can't be blank");
  return false;

}else if(password.length<6){
  alert("Password must be at least 6 characters long.");
  return false;
  }
}
</script>
<body>
```

```
<form name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>
```

56) How to validate email in JavaScript?

```
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
  alert("Please enter a valid e-mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
  return false;
  }
}
```

```
</script>
<body>
<form name="myform"  method="post" action="#" onsubmit="return validateemail();">
Email: <input type="text" name="email"><br/>

<input type="submit" value="register">
</form>
```

57) What is this keyword in JavaScript?

The this keyword is a reference variable that refers to the current object. For example:

```
var address=
{
company:"Javatpoint",
city:"Noida",
state:"UP",
fullAddress:function()
{
```

```
return this.company+" "+this.city+" "+this.state;
}
};
var fetch=address.fullAddress();
document.writeln(fetch);
```

58) What is the requirement of debugging in JavaScript?

JavaScript didn't show any error message in a browser. However, these mistakes can affect the output. The best practice to find out the error is to debug the code. The code can be debugged easily by using web browsers like Google Chrome, Mozilla Firebox.

To perform debugging, we can use any of the following approaches:

- Using console.log() method
- Using debugger keyword

59) What is the use of debugger keyword in JavaScript?

JavaScript debugger keyword sets the breakpoint through the code itself. The debugger stops the execution of the program at the position it is applied. Now, we can start the flow of execution manually. If an exception occurs, the execution will stop again on that particular line.. For example:

```
function display()
{
x = 10;
y = 15;
z = x + y;
debugger;
document.write(z);
document.write(a);
}
display();
```

60) What is the role of a strict mode in JavaScript?

The JavaScript strict mode is used to generates silent errors. It provides "use strict"; expression to enable the strict mode. This expression can only be placed as the first statement in a script or a function. For example:

```
"use strict";
x=10;
console.log(x);
```

61) What is the use of Math object in JavaScript?

The JavaScript math object provides several constants and methods to perform a mathematical operation. Unlike date object, it doesn't have constructors. For example:

```
function display()
{
  document.writeln(Math.random());
}
display();
```

62) What is the use of a Date object in JavaScript?

The JavaScript date object can be used to get a year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

```
function display()
{
  var date=new Date();
var day=date.getDate();
var month=date.getMonth()+1;
var year=date.getFullYear();
document.write("<br>Date is: "+day+"/"+month+"/"+year);
}
display();
```

63) What is the use of a Number object in JavaScript?

The JavaScript number object enables you to represent a numeric value. It may be integer or floating-point. JavaScript number object follows the IEEE standard to represent the floating-point numbers.

```
function display()
{
var x=102;//integer value
var y=102.7;//floating point value
var z=13e4;//exponent value, output: 130000
var n=new Number(16);//integer value by number object
document.write(x+" "+y+" "+z+" "+n);
}
display();
```

64) What is the use of a Boolean object in JavaScript?

The JavaScript Boolean is an object that represents value in two states: true or false. You can create the JavaScript Boolean object by Boolean() constructor.

```
function display()
{
document.writeln(10<20);//true
document.writeln(10<5);//false
}
display();
```

65) What is the use of a TypedArray object in JavaScript?

The JavaScript TypedArray object illustrates an array like a view of an underlying binary data buffer. There is any number of different global properties, whose values are TypedArray constructors for specific element types.

```
function display()
{
var arr1= [1,2,3,4,5,6,7,8,9,10];
      arr1.copyWithin(2) ;
      document.write(arr1);
}
display();
```

66) What is the use of a Set object in JavaScript?

The JavaScript Set object is used to store the elements with unique values. The values can be of any type i.e. whether primitive values or object references. For example:

```
function display()
{
var set = new Set();
set.add("jQuery");
set.add("AngularJS");
set.add("Bootstrap");
```

```
for (let elements of set) {
 document.writeln(elements+"<br>");
}
}
display();
```

67) What is the use of a WeakSet object in JavaScript?

The JavaScript WeakSet object is the type of collection that allows us to store weakly held objects. Unlike Set, the WeakSet are the collections of objects only. It doesn't contain the

arbitrary values. For example:

```
function display()
{
var ws = new WeakSet();
var obj1={};
var obj2={};
ws.add(obj1);
ws.add(obj2);
```

```
//Let's check whether the WeakSet object contains the added object
document.writeln(ws.has(obj1)+"<br>");
document.writeln(ws.has(obj2));
}
display()
```

68) What is the use of a Map object in JavaScript?

The JavaScript Map object is used to map keys to values. It stores each element as key-value pair. It operates the elements such as search, update and delete on the basis of specified key. For example:

```
function display()
{
var map=new Map();
map.set(1,"jQuery");
map.set(2,"AngularJS");
map.set(3,"Bootstrap");
```

```
document.writeln(map.get(1)+"<br>");
document.writeln(map.get(2)+"<br>");
document.writeln(map.get(3));
}
display();
```

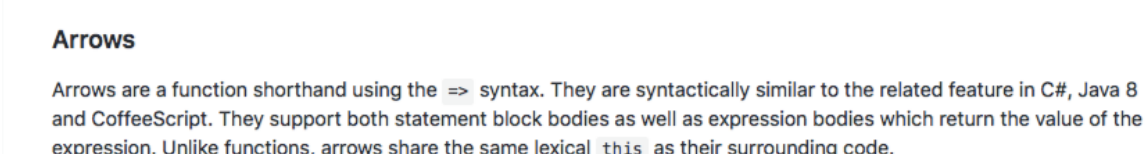
69) What is the use of a WeakMap object in JavaScript?

The JavaScript WeakMap object is a type of collection which is almost similar to Map. It stores each element as a key-value pair where keys are weakly referenced. Here, the keys are objects and the values are arbitrary values. For example:

```
function display()
{
var wm = new WeakMap();
var obj1 = {};
var obj2 = {};
var obj3= {};
```

```
wm.set(obj1, "jQuery");
wm.set(obj2, "AngularJS");
wm.set(obj3,"Bootstrap");
document.writeln(wm.has(obj2));
}
display();
```

FIRST CHEAT SHEET



Arrows

Arrows are a function shorthand using the `=>` syntax. They are syntactically similar to the related feature in C#, Java 8 and CoffeeScript. They support both statement block bodies as well as expression bodies which return the value of the expression. Unlike functions, arrows share the same lexical `this` as their surrounding code.

```
// Expression bodies
var odds = evens.map(v => v + 1);
var nums = evens.map((v, i) => v + i);
var pairs = evens.map(v => ({even: v, odd: v + 1}));

// Statement bodies
nums.forEach(v => {
  if (v % 5 === 0)
    fives.push(v);
});

// Lexical this
var bob = {
  _name: "Bob",
  _friends: [],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " knows " + f));
  }
}
```

More info: MDN Arrow Functions

Classes

ES6 classes are a simple sugar over the prototype-based OO pattern. Having a single convenient declarative form makes class patterns easier to use, and encourages interoperability. Classes support prototype-based inheritance, super calls, instance and static methods and constructors.

```
class SkinnedMesh extends THREE.Mesh {
  constructor(geometry, materials) {
    super(geometry, materials);

    this.idMatrix = SkinnedMesh.defaultMatrix();
    this.bones = [];
    this.boneMatrices = [];
    //...
  }
  update(camera) {
    //...
    super.update();
  }
  get boneCount() {
    return this.bones.length;
  }
  set matrixType(matrixType) {
    this.idMatrix = SkinnedMesh[matrixType]();
  }
  static defaultMatrix() {
    return new THREE.Matrix4();
  }
}
```

More info: MDN Classes

Enhanced Object Literals

Object literals are extended to support setting the prototype at construction, shorthand for `foo: foo` assignments, defining methods, making super calls, and computing property names with expressions. Together, these also bring object literals and class declarations closer together, and let object-based design benefit from some of the same conveniences.

```
var obj = {
    // __proto__
    __proto__: theProtoObj,
    // Shorthand for 'handler: handler'
    handler,
    // Methods
    toString() {
     // Super calls
     return "d " + super.toString();
    },
    // Computed (dynamic) property names
```

```
    // Computed (dynamic) property names
    [ 'prop_' + (() => 42)() ]: 42
};
```

More info: MDN Grammar and types: Object literals

Template Strings

Template strings provide syntactic sugar for constructing strings. This is similar to string interpolation features in Perl, Python and more. Optionally, a tag can be added to allow the string construction to be customized, avoiding injection attacks or constructing higher level data structures from string contents.

```
// Basic literal string creation
`In JavaScript '\n' is a line-feed.`

// Multiline strings
`In JavaScript this is
 not legal.`

// String interpolation
var name = "Bob", time = "today";
`Hello ${name}, how are you ${time}?`

// Construct an HTTP request prefix is used to interpret the replacements and construction
POST`http://foo.org/bar?a=${a}&b=${b}
     Content-Type: application/json
     X-Credentials: ${credentials}
     { "foo": ${foo},
       "bar": ${bar}}`(myOnReadyStateChangeHandler);
```

More info: MDN Template Strings

Destructuring

Destructuring allows binding using pattern matching, with support for matching arrays and objects. Destructuring is fail-soft, similar to standard object lookup `foo["bar"]`, producing `undefined` values when not found.

```
// list matching
var [a, , b] = [1,2,3];

// object matching
var { op: a, lhs: { op: b }, rhs: c }
       = getASTNode()

// object matching shorthand
// binds `op`, `lhs` and `rhs` in scope
var {op, lhs, rhs} = getASTNode()

// Can be used in parameter position
function g({name: x}) {
  console.log(x);
}
g({name: 5})

// Fail-soft destructuring
var [a] = [];
a === undefined;

// Fail-soft destructuring with defaults
var [a = 1] = [];
a === 1;
```

More info: MDN Destructuring assignment

Default + Rest + Spread

Callee-evaluated default parameter values. Turn an array into consecutive arguments in a function call. Bind trailing parameters to an array. Rest replaces the need for `arguments` and addresses common cases more directly.

```
function f(x, y=12) {
  // y is 12 if not passed (or passed as undefined)
  return x + y;
```

```
}
f(3) == 15
```

```
function f(x, ...y) {
  // y is an Array
  return x * y.length;
}
f(3, "hello", true) == 6
```

```
function f(x, y, z) {
  return x + y + z;
}
// Pass each elem of array as argument
f(...[1,2,3]) == 6
```

More MDN info: Default parameters, Rest parameters, Spread Operator

Let + Const

Block-scoped binding constructs. `let` is the new `var`. `const` is single-assignment. Static restrictions prevent use before assignment.

```
function f() {
  {
    let x;
    {
      // okay, block scoped name
      const x = "sneaky";
      // error, const
      x = "foo";
    }
    // error, already declared in block
    let x = "inner";
  }
}
```

More MDN info: let statement, const statement

Iterators + For..Of

Iterator objects enable custom iteration like CLR IEnumerable or Java Iterable. Generalize `for..in` to custom iterator-based iteration with `for..of`. Don't require realizing an array, enabling lazy design patterns like LINQ.

```
let fibonacci = {
  [Symbol.iterator]() {
    let pre = 0, cur = 1;
    return {
      next() {
        [pre, cur] = [cur, pre + cur];
        return { done: false, value: cur }
      }
    }
  }
}

for (var n of fibonacci) {
  // truncate the sequence at 1000
  if (n > 1000)
    break;
  console.log(n);
}
```

Iteration is based on these duck-typed interfaces (using TypeScript type syntax for exposition only):

```
interface IteratorResult {
  done: boolean;
  value: any;
}
interface Iterator {
```

```
  next(): IteratorResult;
}
interface Iterable {
  [Symbol.iterator](): Iterator
}
```

More info: MDN for...of

Generators

Generators simplify iterator-authoring using `function*` and `yield`. A function declared as function* returns a Generator instance. Generators are subtypes of iterators which include additional `next` and `throw`. These enable values to flow back into the generator, so `yield` is an expression form which returns a value (or throws).

Note: Can also be used to enable 'await'-like async programming, see also ES7 `await` proposal.

```
var fibonacci = {
  [Symbol.iterator]: function*() {
    var pre = 0, cur = 1;
    for (;;) {
      var temp = pre;
      pre = cur;
      cur += temp;
      yield cur;
    }
  }
}

for (var n of fibonacci) {
  // truncate the sequence at 1000
  if (n > 1000)
    break;
  console.log(n);
}
```

The generator interface is (using TypeScript type syntax for exposition only):

```
interface Generator extends Iterator {
    next(value?: any): IteratorResult;
    throw(exception: any);
}
```

More info: MDN Iteration protocols

Unicode

Non-breaking additions to support full Unicode, including new Unicode literal form in strings and new RegExp `u` mode to handle code points, as well as new APIs to process strings at the 21bit code points level. These additions support building global apps in JavaScript.

```
// same as ES5.1
"吉".length == 2

// new RegExp behaviour, opt-in 'u'
"吉".match(/./u)[0].length == 2

// new form
"\u{20BB7}"=="吉"=="\uD842\uDFB7"

// new String ops
"吉".codePointAt(0) == 0x20BB7

// for-of iterates code points
for(var c of "吉") {
  console.log(c);
}
```

More info: MDN RegExp.prototype.unicode

Modules

Language-level support for modules for component definition. Codifies patterns from popular JavaScript module loaders (AMD, CommonJS). Runtime behaviour defined by a host-defined default loader. Implicitly async model – no code executes until requested modules are available and processed.

```
// lib/math.js
export function sum(x, y) {
  return x + y;
}
export var pi = 3.141593;
```

```
// app.js
import * as math from "lib/math";
alert("2π = " + math.sum(math.pi, math.pi));
```

```
// otherApp.js
import {sum, pi} from "lib/math";
alert("2π = " + sum(pi, pi));
```

Some additional features include `export default` and `export *`:

```
// lib/mathplusplus.js
export * from "lib/math";
export var e = 2.71828182846;
export default function(x) {
    return Math.log(x);
}
```

```
// app.js
import ln, {pi, e} from "lib/mathplusplus";
alert("2π = " + ln(e)*pi*2);
```

More MDN info: import statement, export statement

Module Loaders

Module loaders support:

- Dynamic loading
- State isolation
- Global namespace isolation
- Compilation hooks
- Nested virtualization

The default module loader can be configured, and new loaders can be constructed to evaluate and load code in isolated or constrained contexts.

```
// Dynamic loading – 'System' is default loader
System.import('lib/math').then(function(m) {
  alert("2π = " + m.sum(m.pi, m.pi));
});

// Create execution sandboxes – new Loaders
var loader = new Loader({
  global: fixup(window) // replace 'console.log'
});
loader.eval("console.log('hello world!');");

// Directly manipulate module cache
System.get('jquery');
System.set('jquery', Module({$: $})); // WARNING: not yet finalized
```

Map + Set + WeakMap + WeakSet

Efficient data structures for common algorithms. WeakMaps provides leak-free object-key'd side tables.

```
// Sets
var s = new Set();
s.add("hello").add("goodbye").add("hello");
s.size === 2;
s.has("hello") === true;

// Maps
var m = new Map();
m.set("hello", 42);
m.set(s, 34);
m.get(s) == 34;

// Weak Maps
var wm = new WeakMap();
wm.set(s, { extra: 42 });
wm.size === undefined

// Weak Sets
var ws = new WeakSet();
ws.add({ data: 42 });
// Because the added object has no other references, it will not be held in the set
```

More MDN info: Map, Set, WeakMap, WeakSet

Proxies

Proxies enable creation of objects with the full range of behaviors available to host objects. Can be used for interception, object virtualization, logging/profiling, etc.

```
// Proxying a normal object
var target = {};
var handler = {
  get: function (receiver, name) {
    return `Hello, ${name}!`;
  }
};

var p = new Proxy(target, handler);
p.world === 'Hello, world!';
```

```
// Proxying a function object
var target = function () { return 'I am the target'; };
var handler = {
  apply: function (receiver, ...args) {
    return 'I am the proxy';
  }
};

var p = new Proxy(target, handler);
p() === 'I am the proxy';
```

There are traps available for all of the runtime-level meta-operations:

```
var handler =
{
  get:...,
  set:...,
  has:...,
  deleteProperty:...,
  apply:...,
  construct:...,
  getOwnPropertyDescriptor:...,
  defineProperty:...,
  getPrototypeOf:...,
  setPrototypeOf:...,
  enumerate:...,
  ownKeys:...,
  preventExtensions:...,
  isExtensible:...
}
```

More info: MDN Proxy

More info: MDN Proxy

Symbols

Symbols enable access control for object state. Symbols allow properties to be keyed by either `string` (as in ES5) or `symbol`. Symbols are a new primitive type. Optional `description` parameter used in debugging - but is not part of identity. Symbols are unique (like gensym), but not private since they are exposed via reflection features like `Object.getOwnPropertySymbols`.

```
var MyClass = (function() {

  // module scoped symbol
  var key = Symbol("key");

  function MyClass(privateData) {
    this[key] = privateData;
  }

  MyClass.prototype = {
    doStuff: function() {
      ... this[key] ...
    }
  };

  return MyClass;
})();

var c = new MyClass("hello")
c["key"] === undefined
```

More info: MDN Symbol

Subclassable Built-ins

In ES6, built-ins like `Array`, `Date` and DOM `Element`s can be subclassed.

Object construction for a function named `Ctor` now uses two-phases (both virtually dispatched):

- Call `Ctor[@@create]` to allocate the object, installing any special behavior
- Invoke constructor on new instance to initialize

The known `@@create` symbol is available via `Symbol.create`. Built-ins now expose their `@@create` explicitly.

```
// Pseudo-code of Array
class Array {
    constructor(...args) { /* ... */ }
    static [Symbol.create]() {
        // Install special [[DefineOwnProperty]]
        // to magically update 'length'
    }
}

// User code of Array subclass
class MyArray extends Array {
    constructor(...args) { super(...args); }
}

// Two-phase 'new':
// 1) Call @@create to allocate object
// 2) Invoke constructor on new instance
var arr = new MyArray();
arr[1] = 12;
arr.length == 2
```

Math + Number + String + Array + Object APIs

Many new library additions, including core Math libraries, Array conversion helpers, String helpers, and Object.assign for copying.

```
Number.EPSILON
Number.isInteger(Infinity) // false
Number.isNaN("NaN") // false
```

```
Math.acosh(3) // 1.762747174039086
Math.hypot(3, 4) // 5
Math.imul(Math.pow(2, 32) - 1, Math.pow(2, 32) - 2) // 2

"abcde".includes("cd") // true
"abc".repeat(3) // "abcabcabc"

Array.from(document.querySelectorAll('*')) // Returns a real Array
Array.of(1, 2, 3) // Similar to new Array(...), but without special one-arg behavior
[0, 0, 0].fill(7, 1) // [0,7,7]
[1, 2, 3].find(x => x == 3) // 3
[1, 2, 3].findIndex(x => x == 2) // 1
[1, 2, 3, 4, 5].copyWithin(3, 0) // [1, 2, 3, 1, 2]
["a", "b", "c"].entries() // iterator [0, "a"], [1,"b"], [2,"c"]
["a", "b", "c"].keys() // iterator 0, 1, 2
["a", "b", "c"].values() // iterator "a", "b", "c"

Object.assign(Point, { origin: new Point(0,0) })
```

More MDN info: Number, Math, Array.from, Array.of, Array.prototype.copyWithin, Object.assign

Binary and Octal Literals

Two new numeric literal forms are added for binary (`b`) and octal (`o`).

```
0b111110111 === 503 // true
0o767 === 503 // true
```

Promises

Promises are a library for asynchronous programming. Promises are a first class representation of a value that may be made available in the future. Promises are used in many existing JavaScript libraries.

```
function timeout(duration = 0) {
    return new Promise((resolve, reject) => {
        setTimeout(resolve, duration);
    })
}

var p = timeout(1000).then(() => {
    return timeout(2000);
}).then(() => {
    throw new Error("hmm");
}).catch(err => {
    return Promise.all([timeout(100), timeout(200)]);
})
```

More info: MDN Promise

Reflect API

Full reflection API exposing the runtime-level meta-operations on objects. This is effectively the inverse of the Proxy API, and allows making calls corresponding to the same meta-operations as the proxy traps. Especially useful for implementing proxies.

```
// No sample yet
```

More info: MDN Reflect

Tail Calls

Calls in tail-position are guaranteed to not grow the stack unboundedly. Makes recursive algorithms safe in the face of unbounded inputs.

```
function factorial(n, acc = 1) {
    'use strict';
    if (n <= 1) return acc;
    return factorial(n - 1, n * acc);
```

```
}

// Stack overflow in most implementations today,
// but safe on arbitrary inputs in ES6
factorial(100000)
```

SECOND CHEAT SHEET

Object.create

Creates a new object with the specified prototype object and properties.

```
o = { }; //is equivalent to: o = Obj
```

Object.defineProperty

Defines a new property directly on an object, or modifies an existing property on an object, and returns the object.

```
Object.defineProperty(obj,
       'answer', {
         value: 42,
         writable: true,
         enumerable: true,
         configurable: true
       });
```

Object.defineProperties

Defines new or modifies existing properties directly on an object, returning the object.

```
var obj = { };
Object.defineProperties(obj, {
  "name": {
    value: true,
    writable: true
  },
  "msg": {
    value: "Hello",
    writable: false
  }//etc.etc.
});
```

Object.getPrototypeOf

Returns the prototype (i.e. the value of the internal [[Prototype]] property) of the specified object.

```
var proto = { };
var obj=Object.create(proto);
Object.getPrototypeOf(obj) === proto
```

Object.keys

Returns an array of a given object's own enumerable properties.

```
var arr = ['x', 'y', 'z'];
console.log(Object.keys(arr)); //['0
```

Object.seal

Prevents any new addition of properties but defined properties can be changed.

```
var obj = {name: 'FooBar'};
Object.seal(obj);
obj.name = 'BarFoo'; //Works delete
```

Object.freeze

Makes an object immutable.

```
var obj = {name: 'FooBar'};
Object.freeze(obj);
obj.name = 'BarFoo';
obj.name //Will still be 'FooBar'
```

Object.preventExtensions

Prevents future extensions to the object and "CONFIGURABLE" is not set to false for all the properites.

```
var obj =  {name: 'FooBar'};
Object.preventExtensions(obj);
obj.name = 'BarFoo';
obj.name //Will still be 'FooBar'
```

Object.isSealed

Checks if a given object is sealed.

```
var obj = {};
Object.seal(obj);
Object.isSealed(obj) //true
```

Object.isFrozen

Checks if a given object is frozen.

```
var obj = {};
Object.freeze(obj);
Object.isFrozen(obj) // true
```

Object.isExtensible

Checks if the given object can be extended.

```
var obj =  {};
Object.preventExtensions(obj);
Object.isExtensible(obj) // false
```

Object.getOwnPropertyDescriptor

Returns a property descriptor for an own property i.e directly on the object and not the prototype chain.

```
Object.getOwnPropertyDescriptor({nam
/*{ value: 'foo', writable: true, en
```

Object.getOwnPropertyNames

Returns an array of all properties (enumerable or not) found directly upon a given object.

```
var obj = { 0: 'f', 1: 'o', 2: 'o' }
Object.getOwnPropertyNames(obj).sort
```

Date.prototype.toISOString

Returns a string in simplified extended ISO format. YYYY-MM-DDTHH:mm:ss.sssZ

```
(new Date('1998-02-01')).toISOString
```

Date.now

Returns the number of milliseconds elapsed since 01 January 1970 00:00:00 UTC.

```
Date.now(); //Something like 1438525
```

Array.isArray

Returns true if an object is an array, false if it is not.

```
Array.isArray(Array.prototype); //tr
Array.isArray(); //true
Array.isArray({}); //false
Array.isArray(null); //false
```

JSON

Contains methods for parsing and creating JSON values.

```
var JSONobj = JSON.stringify({}); //
var obj = JSON.parse(JSONobj); //{}
```

Function.prototype.bind

Creates a new function that, when called, has its exectuion context bound to the provided value.

```
var log = console.log.bind(console);
log('meow') //meow
```

String.prototype.trim

Removes whitespace from both ends of a string.

```
var name = 'foo';
name.trim(); //'foo'
```

Array.prototype.indexOf

Returns the index if found, else returns -1, takes an optional starting index.

```
var name = 'Brendan Eich';
name.indexOf('Eich'); //8
name.indexOf('Brendan', 5); //-1
```

Array.prototype.lastIndexOf

Returns the last index if found or -1 for the specified value, also takes an optional starting index.

```
var city = 'mississippi';
city.lastIndexOf('i'); //10
```

Array.prototype.every

Checks if all the elements of an array passes the specified test.

```
[1, 2, 3].every(function(v, i, a) {
  return v > 3;
});  //false

[1, 2, 3].every(function(v, i, a) {
  return v > 0;
}); //true
```

Array.prototype.some

Checks if any of the elements of an array passes the specified test.

```
[1, 3, 5, 7, 6].some(function(v, i,
  return v%2 === 0;
}); //true
```

Array.prototype.forEach

For each element in an array performs the specified action.

```
['foo', 'bar', 'baz'].forEach(functi
  console.log(v, i);
}); /* foo 0 bar 1 baz 2 */
```

Array.prototype.map

Returns an array that contains the results of a invokation of the function passed to it.

```
[64, 49].map(Math.sqrt); //[8, 7]
```

Array.prototype.filter

Returns an array that meet the condition specified in a callback function.

```
[1, '', true, false].filter(Boolean)
//[1, true]
```

Array.prototype.reduce

Invokes the callback with an accumulator and each value of the array and reduce it to a single value.

```
[0, 1, 2, 3, 4].reduce(function(prev
  return previousValue+currentValue;
}, 10); //20
```

Array.prototype.reduceRight

Similar to reduce, but acts upon the array from right-to-left instead.

```
[0, -1, -2, 5, -6].reduceRight(funct
  return previousValue+currentValue;
}); //-4
```

Getter property in objects

Binds an object property to a function that will be invoked when that property is accessed.

```
foo= { getx() { return42 }};
foo.x; //42
```

Setter property in objects

Binds an object property to a function that will be invoked when that property is been set.

```
var val = 0;
var foo = { set x(v) {
  val = v
}};
foo.x = 42
val; //42
```

Property accessor of strings

Helps to access char in a string by index.

```
"foobar"[2]; //'o'
```

Reserved word property name

Using reserved as property names.

```
{ class: 42 }.class //42
```

ZWSP identifiers

zero-width space identifiers.

```
var _c = 42;
_c; //42
```

ignore leading zeros in parseInt()

Ingoring the leading zero.

```
parseInt('000420'); //420
```

Immutable undefined

Can not mutate undefined.

```
undefined = 42;
```

Arrow functions

Lexical this, shoter functions.

```
let obj = {
```

```
typeof undefined; //'undefined'
```

```
  method: function () {
    return () => this;
  }
};
//Due to lexical scope obj.method()(

let fact = (n) => { return n === 0 ?

let fib = (n) => { return n < 2 ? n
```

Block Scope

Declares a block scope local variable, optionally initializing it to a value.

```
var aboutme = () => {
  {
    var investements = 1;
    const salary = 10;
  }
  console.log(investements,salary);
}
```

class

Syntactical sugar over prototype-based inheritance.

```
class Person {
  constructor(name) {
    this.name = name;
    this.movement = "walks";
  }

  move(meters) {
    console.log(`${this.name} ${this
  }
}

class Hero extends Person {
  constructor(name, movement) {
    this.name = name;
    this.movement = movement;
  }

  move() {
    super.move(500);
  }
}

let clark = new Person("Clark Kent")

let superman = new Hero("Superman",

clark.move(100);
//-> Clark Kent walks 100m.

superman.move();
//-> Superman flies 500m.

/* Make a note of:

class Base {}
class Derived extends Base {}

//Here, Derived.prototype will inher

let parent = {};
class Derived prototype parent {}
*/
```

Computed property names

An expression in brackets `[]`

```
var obj = {
  [foo + bar]: "o_0",
  [foo + baz]: "0_o",
  foo: "foo",
  bar: "bar",
  baz: "baz"
};

console.log(obj.foobar); //o_0
console.log(obj.foobaz); //0_o
```

Default Params

Initialize formal parameters with default values, if no value or undefined is passed.

```
let greet = (msg = "Hello", name = "

greet(); //Hello World!
```

Destructuring

Extract data from arrays or objects.

```
var {foo, bar} = {foo: "lorem", bar:
//foo => lorem and bar => ipsum
```

Direct Proxy

Define custom behavior for fundamental operations of an object.

```
let NegativeIndices = (array) => {
  return new Proxy(array, {
    get: (receiver, name) => {
      let index;
      console.log('Proxy#get', array
      index = parseInt(name);
      if (!isNaN(index) && index < 0
        array[array.length + index];
      } else {
        return array[name];
      }
    }
  });
};

/*
* Negative array indices:
* array = NegativeIndices [4, 420, 4
* array[-1] is 42
*/
```

for-of loop

Loop over Iterator objects.

```
for (let element of [1, 2, 3]) {
  console.log(element);
}
```

Generators

The function* declaration defines a generator function, which returns a Generator object.

```
function *Counter(){
  var n = 0;
```

Map

Map object is a simple and effective key/value data-structure.

```
let m = new Map();
m.set('answer', 42);
```

```
  while(1 < 2) {
      yield n;
      ++n;
  }
}

var CountIter = new Counter();

CountIter.next(); //{value: 0, done:
CountIter.next(); //{value: 1, done:
CountIter.next(); //{value: 2, done:
```

```
m.get('answer'); //42
m.has('answer'); //true
m.delete('answer');// true
m.has('answer'); //false

m.set(keyFunc,() => "foo");
m.get(keyFunc)(); //"foo"
```

modules

Module format common to CommonJS and AMD.

```
/* In math.js */
export function div(x, y) {
  return x / y;
}
export var pi = 3.141593;

//In index.js
import {div, pi} from math;
```

Better Object Literal

Better as in the example.

```
var greet = {
  __proto__: theProtoObj,
  handler, //Instead of handler: han
  world: () => "Hello World!",
  toString() {
    return "Results: " + super.toStr
  }
};
```

property-method-assignment

Method syntax is supported in object initializers.

```
let person = {
  get name() {
    return this._name;
  },
  set name(val){
    console.log("Setting name: " + v
    this._name = val;
  }
};

/*
> person.name = "Hemanth"
"Hemanth"
"Setting name: Hemanth"

> person.name
```

Rest params

Variable number of arguments without using the arguments object.

```
let sortRestArgs = (...theArgs) => t

console.log(sortRestArgs(5,2,7,1)) /
```

Set

Store unique values of any type.

```
var cards = new Set();
cards.add('♠');
cards.add('♥');
cards.add('♦');
cards.add('♣');

cards.has('♠'); //true

cards.has('joker'); //false

cards.size; //4

cards.forEach((card) => console.log(

/*
Would log:
♠
♥
♦
♣
*/

cards.add('♣');

cards.size //Still four as ♣ was alr
```

Spread operator

Expanded in places with `...` for arguments or multiple elements.

```
var nodeList = document.querySelecto
var array = [...nodeList];
```

Symbol

Unique and immutable data type.

```
var Cat = (function() {
    var nameSymbl = Symbol('name');

    function Cat(name) {
        this[nameSymbl] = name;
    }

    Cat.prototype.getName = function
        return this[nameSymbl];
    };

    return Cat;
}());

var c = new Cat('milly');
console.log("Cat's name: " + c.getNa
delete c.name; //Even after deleting
console.log("Cat's name is still: "
```

Tail recursion

Tail Calls, Optimization.

```
let factorial = (n, acc = 1) => {
    if (n <= 1) return acc;
    return factorial(n - 1, n * acc)
}
//NO S.O!
factorial(133333337);
```

Template Literals

Better string formatting capabilities.

```
var First = "Hemanth";
var Last = " HM";
`${First} + ${Last} = ${First + Last
//"Hemanth + HM = Hemanth HM"
```

Unicode in Regex

Unicode aware regex.

```
var string = 'foo𠮷bar';
var match = string.match(/foo(.)bar/
console.log(match[1]);
//'𠮷'
```

WeakMap

key/value pairs, keys are objects and the values can be arbitrary values, references to key objects are held "weakly".

```
var wm = new WeakMap();

wm.set('life'); //TypeError: Invalid

wm.set('life', 'life'.length) //Type

var wmk = {};

wm.set(wmk,'life');

wm.get(wmk); //"life"

wm.has(wmk); //true

wm.delete(wmk); //true

wm.has(wmk); //false
```

WeakSet

Store weakly held objects in a collection.

```
var ws = new WeakSet();
var foo = {};
var bar = {};

ws.add(window);
ws.add(foo);

ws.has(window); //true
ws.has(bar); //false, bar has not be

ws.delete(window); //removes window
ws.has(window); //false, window has

ws.clear(); //empty the whole WeakSe
```

Exponentiation Operator

Performs exponential calculation on operands. Same algorithm as Math.pow(x, y).

```
let cubed = x => x ** 3;
cubed(2) //8;
```

Async functions

Deferred generators.

```
function wait(t) {
    return new Promise((r) => setTim
}
async function asyncMania() {
    console.log("1");
    await wait(1000);
    console.log("2");
}
asyncMania()
.then(() => alert("3"));
//logs: 1 2 3
```

Object Observe

Asynchronously observing the changes to an object.

```
var obj = {};
Object.observe(obj, function(changes
    console.log(changes);
});
obj.name = "hemanth";
//Would log -> [ { type: 'new', obje
```

Object.getOwnPropertyDescriptors

Returns a property descriptor for an own property.

```
//Creating a shallow copy.
var shallowCopy = Object.create(
    Object.getPrototypeOf(originalOb
    Object.getOwnPropertyDescriptors
);
```

Object.values

Get all the values of the object as an array.

```
var person = { fname: "Hemanth", lna
Object.values(person);
//["Hemanth","HM","Earth","Human"]
```

Object.entries

Returns a Array of arrays of key,value pairs.

```
var person = { fname: "Hemanth", lna
Object.entries(person);
//[["fname","Hemanth"],["lname","HM"
```

Array.prototype.includes

Determines whether an array includes a certain element or not.

```
[1, 2, 3].includes(3, 0, 7); //true
[1, 2, NaN].includes(NaN); //true
[0,+1,-1].includes(42); //false
```

Typed Objects

Portable, memory-safe, efficient, and structured access to contiguously allocated data.

```
var Point = new StructType({
    x: int32,
    y: int32
});
var point = new Point({
    x: 42,
    y: 420
});
```

Trailing commas in function syntax

Trailing commas in parameter and argument lists.

```
var meow = function (cat1, cat2,) {}
Math.max(4,2,0,);
```

Class properties

Properties of class.

```
class Cat {
    name = 'Garfield';
    static says = 'meow';
}
new Cat().name; //Garfield
Cat.says; //meow
```

Map.prototype.toJSON

toJSON for Maps.

```
var myMap = new Map();
myMap.set(NaN, "not a number");
console.log(myMap.toJSON()); //{"NaN
```

Set.prototype.toJSON

toJSON for Sets.

```
var mySet = new Set();
mySet.add(NaN);
mySet.add(1);
console.log(mySet.toJSON()) //{"1":1
```

String.prototype.at

Object spread properties

String.prototype.padLeft

String containing the code point at the given position.

```
'abc𝌆def'.at(1) //'b'
'abc𝌆def'.at(3) //'𝌆'
```

Spread properties for object destructuring assignment.

```
let info = {fname, lname, ...rest};
info; //{ fname: "Hemanth", lname: "
```

left justify and pad strings.

```
"hello".padLeft(4); //"hello"
"hello".padLeft(20); //"hello"
"hello".padLeft(20, '1234') //"hello
```

String.prototype.padRight

Right justify and pad strings.

```
"hello".padRight(4); //"hello"
"hello".padRight(20; //hello"
"hello".padRight(20, '1234'); //"123
```

String.prototype.trimLeft

Left trim strings.

```
' \t \n LeftTrim  \t\n'.trimLeft();
//LeftTrim  \t\n
```

String.prototype.trimRight

Right trim strings.

```
' \t \n TimeRight   \t\n'.trimLeft()
//\t \n LeftRight
```

Regexp.escape

Escapes any characters that would have special meaning in a regular expression.

```
RegExp.escape("(*.*)"); //"(*.*)"
RegExp.escape("｡^･ɪ･^｡") //"｡^･ɪ･^｡"
RegExp.escape("😊 *_* +_+ ... 👍");
```

Bind Operator

:: Function binding and method extraction

```
let log = (level, msg) => ::console[
import { map, takeWhile, forEach } f
getPlayers()
    ::map(x => x.character())
    ::takeWhile(x => x.strength > 10
    ::forEach(x => console.log(x));
```

Reflect.Realm

TDB

```
TBD
```

THIRD CHEAT SHEET

Constants

```
> const EULER = 2.7182818284
> EULER = 13
> EULER
> 2.7182818284
```

Warning! If array or object, the reference is kept constant. If the constant is a reference to an object, you can still modify the content, but never change the variable.

```
> const CONSTANTS = []
> CONSTANTS.push(EULER)
> CONSTANTS
> [ 2.7182818284 ]
> CONSTANTS = { 'euler': 2.7182818284 }
> CONSTANTS
> [ 2.7182818284 ]
```

Binary, Octal and Hex Notation

```
> 0b1001011101 // 605
> 0o6745 // 3557
> 0x2f50a // 193802
```

New Types

Symbols, Maps, WeakMaps and Sets

let vs var

```
> var average = 5
> var average = (average + 1) / 2
> average
> 3
> let value = 'hello world'
> let value = 'what is new'
// -> throws TypeError: Identifier 'value' has
already been declared
```

Be aware of Temporal Dead Zones:

```
> console.log(val) // -> 'undefined'
> var val = 3
> console.log(val) // -> 3
```

Because it's equivalent to:

```
> var val
> console.log(val)
> val = 3
> console.log(val)
```

Variables declared with "let/const" do not get hoisted:

```
> console.log(val)
// -> Throws ReferenceError
> let val = 3
> console.log(val) // -> 3
```

Arrow Function

```
> setTimeout(() => {
…  console.log('delayed')
… }, 1000)
```

Equivalent with Anonymous Function

```
> setTimeout(function () {
…   console.log('delayed')
… }.bind(this), 1000)
```

New Scoped Functions

```
> {
… let cue = 'Luke, I am your father'
… console.log(cue)
… }
> 'Luke, I am your father'
```

Equivalent with Immediately Invoked Function Expressions (IIFE)

```
> (function () {
… var cue = 'Luke, I am your father'
… console.log(cue) // 'Luke, I am –
… }())
> console.log(cue) // Reference Error
```

Object Notation Novelties

```
// Computed properties
> let key = new Date().getTime()
> let obj = {  [key]: "value" }
> obj
> { '1459958882881': 'value' }


// Object literals
balloon = { color, size };

// Same as
balloon = {
  color: color,
  size: size
}

// Better method notations
obj = {
    foo (a, b) { … },
    bar (x, y) { … }
}
```

String Interpolation, Thanks to Template Literals

```
> const name = 'Tiger'
> const age = 13
> console.log(`My cat is named ${name} and is
${age} years old.`)
> My cat is named Tiger and is 13 years old.

// We can preserve newlines…
let text = ( `cat
dog
nickelodeon`
)
```

Default Params

```
> function howAreYou (answer = 'ok') {
 console.log(answer) // probably 'ok'
}
```

Promises

```
new Promise((resolve, reject) => {
  request.get(url, (error, response,
  body) => {
    if (body) {
        resolve(JSON.parse(body));
      } else {
        resolve({});
      }
  })
}).then(() => { ... })
.catch((err) => throw err)

// Parallelize tasks
Promise.all([
   promise1, promise2, promise3
]).then(() => {
   // all tasks are finished
})
```

Classes, Inheritance, Setters, Getters

```
class Rectangle extends Shape {
  constructor (id, x, y, w, h) {
    super(id, x, y)
    this.width = w
    this.height = h
  }
  // Getter and setter
  set width (w) { this._width = w }
  get width () { return this._width }
}

class Circle extends Shape {
  constructor (id, x, y, radius) {
    super(id, x, y)
    this.radius = radius
  }
  do_a(x) {
    let a = 12;
    super.do_a(x + a);
  }
  static do_b() { ... }
}
Circle.do_b()
```

Destructuring Arrays

```
> let [a, b, c, d] = [1, 2, 3, 4];
> console.log(a);
> 1
> b
> 2
```

Destructuring Objects

```
> let luke = {  occupation: 'jedi',
 father: 'anakin' }
> let {occupation, father} = luke
> console.log(occupation, father)
> jedi anakin
```

Spread Operator

```
// Turn arrays into comma separated
// values and more
> function logger (...args) {
 console.log('%s arguments',
    args.length)
 args.forEach(console.log)
 // arg[0], arg[1], arg[2]
}
```

Or Do a Better Push

```
> let arr = [1, 2, 3]
> [...arr, 4, 5, 6]
> [1, 2, 3, 4, 5, 6]
```

...Go Destructuring Like a Boss

```
> const [ cat, dog, ...fish ] = [
'schroedinger',  'Laika', 'Nemo', 'Dori']
> fish // -> ['Nemo', 'Dori']
```

...And Destructuring in the Future ES7

```
{a, b, ...rest} = { a:1, b:2, c:3, d:4 }
```

Async ES7

```
async function schrodinger () {
return new Promise((resolve, reject)
  => {
   const result = Math.random > 0.5
   setTimeout(() => {
     return result ? resolve('alive')
     : reject('dead')
   })
  })
}
```

Await ES7

```
try {
   console.log(await schrodinger())
   // -> 'alive'
} catch (err) {
   console.log(err)
   // -> 'dead'
}
```

Export ES7

```
export function sumTwo (a, b) {
    return a + b;
}
export const EULER = 2.7182818284
let stuff = { sumTwo, EULER }
export { stuff as default }
```

Importing ES7

```
import React from 'react'
import { EULER } from './myexports'
import * as stuff from './myexports'
// equivalent to
import stuff from './myexports'
// { sumTwo, EULER }
```

Generators

They return a objects that implement an iteration protocol. i.e. it has a next() method that returns { value: < some value>, done: <true or false> }.

```
function* incRand (max) { // Asterisk defines this as a generator
  while (true) {
    // Pause execution after the yield, resume
    // when next(<something>) is called
    // and assign <something> to x
    let x = yield Math.floor(Math.random() * max + 1);
    max += x;
  }
}
```

```
> var rng = incRand(2) // Returns a generator object
> rng.next() // { value: <between 1 and 2>, done: false }
> rng.next(3) // as above, but between 1 and 5
> rng.next() // NaN since 5 + undefined results in NaN
> rng.next(20) // No one expected NaN again?
> rng.throw(new Error('Unrecoverable generator state.'))
// Will be thrown from yield
```