JavaScript

document.write(“hello”); // will write hello to the document body tag,

BOM – browser object model, it gives all the info about the browser, it has some default objects created, like **window, history etc..** these are created by browser only, no need to create manually

DOM – document object model, it gives all the info about the HTML which is displaying on browser.

function add(){

// add function declaration.

}

var add = function(){

// add function expression

}

Self-invoking function

(function(){

//gets called automatically on load

})();

Datatypes in JavaScript :

|  |  |
| --- | --- |
| 1. Primitive datatypes:    1. String    2. Number    3. Boolean    4. Undefined    5. Null | 1. Non Primitive Datatypes    1. Object    2. Array    3. RegExp |

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

document.getElementById(‘selector’).innerHTML(‘<p>dynamic element</p>’); can add the element dynamically

document.getElementById(‘selector’).innerText = “dynamic text” // can add the text dynamically to selected element

There are 3 ways to create objects.

1. By object literal

emp={id:102,name:"Shyam Kumar",salary:40000}

1. By creating instance of Object directly (using new keyword)

var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;

1. By using an object constructor (using new keyword)

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

}

e=new emp(103,"Vimal Jaiswal",30000);

and emp.id , emp.name, emp.salary will get the values

How to create array in JavaScript?

1. By array literal

var emp=["Sonoo","Vimal","Ratan"];

1. By creating instance of Array

var emp = new Array();

emp[0] = "Arun";

emp[1] = "Varun";

emp[2] = "John";

1. By using an Array constructor

var emp=new Array("Jai","Vijay","Smith");

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

"10"+20+30 = 102030 String

10+20+"30" = 3030 String

Undefined value: A value that is not defined and has no keyword is known as undefined value. For example: int number;//Here, number has undefined value.

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

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

Here if we see variable number is undefined value

But if we check “numbers” is not defined.

Because numbers variable is not defined.

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

"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.

3 types of popups : Alert Box , Confirm Box, Prompt Box

navigator.appVersion gives OS of the system

code to submit form on 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>

Get form values

var name=document.myform.name.value;

var password=document.myform.password.value;

// myform is the name attribute of the form

What is the mul function to get below output:

console.log(mul(2)(3)(4)); // output : 24

console.log(mul(4)(3)(4)); // output : 48

function mul (x) {

return function (y) { // anonymous function

return function (z) { // anonymous function

return x \* y \* z;

};

};

}

How to empty an array in js

arrayList.length = 0;

or

arrayList.splice(0, arrayList.length);

to check an object is array or not

Object.prototype.toString.call( arrayList ) === '[object Array]' // this will return true

If its jQuery

$.isArray(arrayList) or Array.isArray(arrayList) // returns true

|  |  |  |
| --- | --- | --- |
| var x = 1;  var output = (function(){  delete x;  return x;  })();    console.log(output); // 1  The delete operator is used to delete the property of an object. Here x is not an object, but rather it's the global variable of type number | var output = (function(x){  delete x;  return x;  })(0);    console.log(output); // 0  The delete operator is used to delete properties from an object. Here x is not an object but a local variable. delete operators don't affect local variables. | var x = { foo : 1};  var output = (function(){  delete x.foo;  return x.foo;  })();    console.log(output); //undefined  Here, x is an object which has the property foo, and as it is a self-invoking function, we will delete the foo property from object x. After doing so, when we try to reference a deleted property foo, |

var trees = ["redwood","bay","cedar","oak","maple"];

delete **trees**[3];

When you run the code above and type console.log(trees); into your Chrome developer console, you will get ["redwood", "bay", "cedar", undefined × 1, "maple"]. When you run the code in Firefox's browser console, you will get ["redwood", "bay", "cedar", undefined, "maple"]. Thus, it's clear that the Chrome browser has its own way of displaying uninitialised indexes in arrays. However, when you check

trees[3] === undefined in both browsers, you will get similar output as true.

Note: Please remember you do not need to check for the uninitialised index of array in trees[3] === 'undefined × 1', as it will give you an error. 'undefined × 1' is just way of displaying an array's uninitialised index in Chrome.

And still the trees.length is 5,

only the delete operator removes an array element, that deleted element is not longer present in array. In place of value at deleted index undefined x 1 in chrome and undefined is placed at the index.

var bar = true;

console.log(bar + 0); //1

console.log(bar + "xyz"); // truexyz as string

console.log(bar + true); // 2

console.log(bar + false); //1

Number + Number -> Addition

Boolean + Number -> Addition // if Boolean is true then consider it as 1, if its false consider it as 0

Number + String -> Concatenation

String + Boolean -> Concatenation

String + String -> Concatenation

var z = 1, y = z = typeof y;

console.log(y); // undefined

The output would be undefined. According to the associativity rule, operators with the same precedence are processed based on the associativity property of the operator. Here, the associativity of the assignment operator is Right to Left, so typeof y will evaluate first , which is undefined. It will be assigned to z, and then y would be assigned the value of z and then z would be assigned the value 1.

|  |  |  |
| --- | --- | --- |
| var foo = function bar(){ return 12; };  typeof bar(); // Reference Error  bar's reference variable points to anonymous function | var bar = function(){ return 12; };  typeof bar();  // 12 | function bar(){ return 12; };  typeof bar();  //12 |

**setTimeout(function, delay)** function is used to start a timer that calls a particular function after the mentioned delay.

**setInterval(function, delay)** function is used to repeatedly execute the given function in the mentioned delay and only halts when cancelled

**clearInterval(id)** function instructs the timer to stop.

What is the use of Void(0)

**Void(0)** is used to prevent the page from refreshing and parameter “zero” is passed while calling.

**Void(0)** is used to call another method without refreshing the page

the Shift() method works at the start of the array and returns it

the pop() method take the last element off of the given array and returns it

The array on which pop/shift are called is then altered.

var cloths = [“Shirt”, “Pant”, “TShirt”];

cloths.pop();

The push() method is used to add or append one or more elements to the end of an Array.

Unshift() method is like push method which works at the beginning of the array. This method is used to prepend one or more elements to the beginning of the array.

**Break** statement exits from the current loop.

**Continue** statement continues with next statement of the loop.

Typeof’ is an operator which is used to return a string description of the type of a variable.

Var a =”phani”;

a.typeOf() // string

The **window.onload()** function is not run until all the information on the page is loaded. This leads to a substantial delay before any code is executed.

**onDocumentReady** loads the code just after the DOM is loaded. This allows early manipulation of the code.

JQuery – latest is 3.2.1 v

var jq = jQuery.**noConflict**();

we can use jq(“#id”) in place of $

var arr = ["Goergie", "Johnson", "Agile", "Harrison", "Gaurav"];

$.each(arr, function(index, value) {

alert('Position is : ' + index + ' And Value is : ' + value);

});

<input id="check1" checked="checked" type="checkbox" />

.attr('checked') //returns checked

.prop('checked') //returns true

.is(':checked') //returns true

Jquery UI : Draggable , Droppable, Resizable, Selectable, Sortable

Effects in Jquery :

animate( params, [duration, easing, callback] )

fadeIn( speed, [callback] )

fadeOut( speed, [callback] )

fadeTo( speed, opacity, callback )

stop( [clearQueue, gotoEnd ])

**$(window).load** is an event that fires when the DOM and all the content (everything) on the page is fully loaded. This event is fired after the **$(document).ready**. $(window).load event is fired once the DOM and all the CSS, images and frames are fully loaded.

jQuery **bind()** method attaches an event handler to elements,

**unbind()** detaches an existing event handler from elements.

Remove an HTML Element dynamically : empty(), remove(), html()

Jquery filters : .eq(), .first(), .last(), .filter(), .has(), .not()

$(document).ready(function()) is a jQuery event that fires as soon as the DOM is fully loaded and ready to be manipulated by script. This is the earliest point in the page load process where the script can safely access elements in the page's HTML DOM. This event is fired before all the images and CSS are fully loaded.

Slice and splice will return a subset of array from main set.

cloths = ["1","2","3","4"]

cloths.slice(1,3) // ["2", "3"] slice is including start index excluding end index,

cloths = ["1","2","3","4"]

cloths.splice(1.3) // ["2", "3", "4"] splice is including start and end index

**adding css from jquery**

$("p").css({"background-color": "yellow", "font-size": "200%"});

We can have any number **of document.ready() function on the same page.**

body.onload() function is called when everything gets loaded on the page that includes DOM, images and all associated resources of the page.

document.ready() function is called as soon as DOM is loaded

How to load jQuery locally when CDN fails

<script type="text/javascript">

if (typeof jQuery == 'undefined')

{

document.write(unescape("%3Cscript src='Scripts/jquery.1.9.1.min.js' type='text/javascript'%3E%3C/script%3E"));

}

</script>

$("div.parent") // All the div element with class parent.

$("p#elmID .myCssClass"); // all the p tags with id eImID and class myCsscClass

But the orde of execution is jQuery will first find all the elements with class ".myCssClass" and after that it will reject all the other elements which are not in "p#elmID".

(selector).animate({styles},speed,easing,callback)

.empty(): This method removes all the child element of the matched element

.remove(): Use .remove() when you want to remove the element itself, as well as everything inside it. In addition to the elements themselves, all bound events and jQuery data associated with the elements are removed.

.detach(): This method is the same as .remove(), except that .detach() keeps all jQuery data associated with the removed elements. This method is useful when removed elements are to be reinserted into the DOM at a later time.

.bind() vs .live() vs .delegate() vs .on()

bind() : to bind events to elements, but cannot work on dynamic elements, bind() only attach events to the current elements not future element

live() : It works for dynamically added elements or future elements., but due to its poor performance, this method is deprecated from jQuery 1.7

delegate():method attaches one or more event handlers for specified elements that are children of selected elements, and specifies a function to run when the events occur.

.on(): is from JQuery 1.7 version,This method provides all the goodness of previous 3 methods and it brings uniformity for attaching event handlers.

$('#dvText').clone().appendTo('body'); // clone and appendTo // clones element and append to body without events associated to source element

$('#dvClickme').clone(**true**).appendTo('body'); // along with events

**event.preventDefault():** Stops the default action of an element from happening.

**event.stopPropagation():** Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event. For example, if there is a link with a click method attached inside of a DIV or FORM that also has a click method attached, it will prevent the DIV or FORM click method from firing

**return false** will do both.

**event.stopPropagation()** allows other handlers on the same element to be executed, while **event.stopImmediatePropagation()** prevents future events also from running.

$("#btnDummy").one("click", function() {

alert("This will be displayed only once.");

}); // one() method add event to handler only once

The .**finish**() method is similar to .**stop**(true, true) in that it clears the queue and the current animation jumps to its end value. It differs, however, in that .finish() also causes the CSS property of all queued animations to jump to their end values, as well.

JQuery plugin example:

|  |  |
| --- | --- |
| In abc.js  jQuery.fn.warning = function() {  return this.each(function() {  alert('Tag Name:"' + $(this).prop("tagName") + '".');  });  };  Syntax : jQuery.fn.methodName = methodDefinition; | In 123.js , import abc.js  $("div").warning();  $("p").warning(); |

Jquery Ajax:

*// Using the core $.ajax() method*

$.ajax({

*// The URL for the request*

url: "post.php",

*// The data to send (will be converted to a query string)*

data: {

id: 123

},

*// Whether this is a POST or GET request*

type: "GET",

*// The type of data we expect back*

dataType : "json",

})

*// Code to run if the request succeeds (is done);*

*// The response is passed to the function*

.done(**function**( json ) {

$( "<h1>" ).text( json.title ).appendTo( "body" );

$( "<div class=\"content\">").html( json.html ).appendTo( "body" );

})

*// Code to run if the request fails; the raw request and*

*// status codes are passed to the function*

.fail(**function**( xhr, status, errorThrown ) {

alert( "Sorry, there was a problem!" );

console.log( "Error: " + errorThrown );

console.log( "Status: " + status );

console.dir( xhr );

})

*// Code to run regardless of success or failure;*

.always(**function**( xhr, status ) {

alert( "The request is complete!" );

});

There are many, many options for the $.ajax() method, which is part of its power. Ex:

Async: Set to false if the request should be sent synchronously. Defaults to true. Note that if you set this option to false, your request will block execution of other code until the response is received.

cache: Whether to use a cached response if available. Defaults to true for all dataTypes except "script" and "jsonp". When set to false, the URL will simply have a cachebusting parameter appended to it.

done: A callback function to run if the request succeeds. The function receives the response data (converted to a JavaScript object if the dataType was JSON), as well as the text status of the request and the raw request object.

fail: A callback function to run if the request results in an error. The function receives the raw request object and the text status of the request.

always: A callback function to run when the request is complete, regardless of success or failure. The function receives the raw request object and the text status of the request.

context: The scope in which the callback function(s) should run (i.e. what this will mean inside the callback function(s)). By default, this inside the callback function(s) refers to the object originally passed to $.ajax().

data: The data to be sent to the server. This can either be an object or a query string, such as foo=bar&amp;baz=bim.

dataType: The type of data you expect back from the server. By default, jQuery will look at the MIME type of the response if no dataType is specified.

jsonp: The callback name to send in a query string when making a JSONP request. Defaults to "callback".

timeout: The time in milliseconds to wait before considering the request a failure.

traditional: Set to true to use the param serialization style in use prior to jQuery 1.4.

type :The type of the request, "POST" or "GET". Defaults to "GET". Other request types, such as "PUT" and "DELETE" can be used, but they may not be supported by all browsers.

url : The URL for the request.

The url option is the only required property of the $.ajax() configuration object; all other properties are optional. This can also be passed as the first argument to $.ajax(), and the options object as the second argument.

If we don’t want to use complete ajax and there is no need of handling errors in response we can directly use .get, .post, .getScript, .getJSON

Ex: *// Using jQuery's Ajax convenience methods*

*// Get plain text or HTML*

$.get( "/users.php", {

userId: 1234

}, **function**( resp ) {

console.log( resp ); *// server response*

});

*// Add a script to the page, then run a function defined in it*

$.getScript( "/static/js/myScript.js", **function**() {

functionFromMyScript();

});

*// Get JSON-formatted data from the server*

$.getJSON( "/details.php", **function**( resp ) {

*// Log each key in the response data*

$.each( resp, **function**( key, value ) {

console.log( key + " : " + value );

});

});

There is another method in ajax .load

$.fn.load : The .load() method is unique among jQuery’s Ajax methods in that it is called on a selection. The .load() method fetches HTML from a URL, and uses the returned HTML to populate the selected element(s). In addition to providing a URL to the method, you can optionally provide a selector; jQuery will fetch only the matching content from the returned HTML.

Ex:

*// Using .load() to populate an element*

$( "#newContent" ).load( "/foo.html" );

*// Using .load() to populate an element based on a selector*

$( "#newContent" ).load( "/foo.html #myDiv h1:first", **function**( html ) {

alert( "Content updated!" );

});

Jquery Filters:

.eq() , .first(), .last(), .filter(), .has(), .not()

Jquery Events:

bind(), unbind(), blur(), off(), hover(), on(), one(), ready(), trigger() etc.