1. **What are the possible ways to create objects in JavaScript**

There are many ways to create objects in javascript as below

* 1. **Object constructor:**

The simplest way to create an empty object is using the Object constructor. Currently this approach is not recommended.

var object = new Object();

* 1. **Object's create method:**

The create method of Object creates a new object by passing the prototype object as a parameter

var object = Object.create(null);

* 1. **Object literal syntax:**

The object literal syntax is equivalent to create method when it passes null as parameter

var object = {};

* 1. **Function constructor:**

Create any function and apply the new operator to create object instances,

function Person(name){

this.name=name;

this.age=21;

}

var object = new Person("Sudheer");

* 1. **Function constructor with prototype:**

This is similar to function constructor but it uses prototype for their properties and methods,

function Person(){}

Person.prototype.name = "Sudheer";

var object = new Person();

This is equivalent to an instance created with an object create method with a function prototype and then call that function with an instance and parameters as arguments.

function func {};

new func(x, y, z);

**(OR)**

// Create a new instance using function prototype.

var newInstance = Object.create(func.prototype)

// Call the function

var result = func.call(newInstance, x, y, z),

// If the result is a non-null object then use it otherwise just use the new instance.

console.log(result && typeof result === 'object' ? result : newInstance);

* 1. **ES6 Class syntax:**

ES6 introduces class feature to create the objects

class Person {

constructor(name) {

this.name = name;

}

}

var object = new Person("Sudheer");

* 1. **Singleton pattern:**

A Singleton is an object which can only be instantiated one time. Repeated calls to its constructor return the same instance and this way one can ensure that they don't accidentally create multiple instances.

var object = new function(){

this.name = "Sudheer";

}

1. **What is a prototype chain**

**Prototype chaining** is used to build new types of objects based on existing ones. It is similar to inheritance in a class based language.

The prototype on object instance is available through **Object.getPrototypeOf(object)** or **proto** property whereas prototype on constructors function is available through **Object.prototype**.

[Screenshot](https://github.com/sudheerj/javascript-interview-questions/blob/master/images/prototype_chain.png)

1. **What is the difference between Call, Apply and Bind**

The difference between Call, Apply and Bind can be explained with below examples,

**Call:** The call() method invokes a function with a given this value and arguments provided one by one

var employee1 = {firstName: 'John', lastName: 'Rodson'};

var employee2 = {firstName: 'Jimmy', lastName: 'Baily'};

function invite(greeting1, greeting2) {

console.log(greeting1 + ' ' + this.firstName + ' ' + this.lastName+ ', '+ greeting2);

}

invite.call(employee1, 'Hello', 'How are you?'); // Hello John Rodson, How are you?

invite.call(employee2, 'Hello', 'How are you?'); // Hello Jimmy Baily, How are you?

**Apply:** Invokes the function with a given this value and allows you to pass in arguments as an array

var employee1 = {firstName: 'John', lastName: 'Rodson'};

var employee2 = {firstName: 'Jimmy', lastName: 'Baily'};

function invite(greeting1, greeting2) {

console.log(greeting1 + ' ' + this.firstName + ' ' + this.lastName+ ', '+ greeting2);

}

invite.apply(employee1, ['Hello', 'How are you?']); // Hello John Rodson, How are you?

invite.apply(employee2, ['Hello', 'How are you?']); // Hello Jimmy Baily, How are you?

**bind:** returns a new function, allowing you to pass any number of arguments

var employee1 = {firstName: 'John', lastName: 'Rodson'};

var employee2 = {firstName: 'Jimmy', lastName: 'Baily'};

function invite(greeting1, greeting2) {

console.log(greeting1 + ' ' + this.firstName + ' ' + this.lastName+ ', '+ greeting2);

}

var inviteEmployee1 = invite.bind(employee1);

var inviteEmployee2 = invite.bind(employee2);

inviteEmployee1('Hello', 'How are you?'); // Hello John Rodson, How are you?

inviteEmployee2('Hello', 'How are you?'); // Hello Jimmy Baily, How are you?

Call and apply are pretty interchangeable. Both execute the current function immediately. You need to decide whether it’s easier to send in an array or a comma separated list of arguments. You can remember by treating Call is for **comma** (separated list) and Apply is for **Array**.

Whereas Bind creates a new function that will have this set to the first parameter passed to bind().

1. **What is JSON and its common operations**

**JSON** is a text-based data format following JavaScript object syntax, which was popularized by Douglas Crockford. It is useful when you want to transmit data across a network and it is basically just a text file with an extension of .json, and a MIME type of application/json

**Parsing:** Converting a string to a native object

JSON.parse(text)

**Stringification:** converting a native object to a string so it can be transmitted across the network

JSON.stringify(object)

1. **What is the purpose of the array slice method**

The **slice()** method returns the selected elements in an array as a new array object. It selects the elements starting at the given start argument, and ends at the given optional end argument without including the last element. If you omit the second argument then it selects till the end.

Some of the examples of this method are,

let arrayIntegers = [1, 2, 3, 4, 5];

let arrayIntegers1 = arrayIntegers.slice(0,2); // returns [1,2]

let arrayIntegers2 = arrayIntegers.slice(2,3); // returns [3]

let arrayIntegers3 = arrayIntegers.slice(4); //returns [5]

**Note:** Slice method won't mutate the original array but it returns the subset as a new array.

1. **What is the purpose of the array splice method**

The **splice()** method is used either adds/removes items to/from an array, and then returns the removed item. The first argument specifies the array position for insertion or deletion whereas the optional second argument indicates the number of elements to be deleted. Each additional argument is added to the array.

Some of the examples of this method are,

let arrayIntegersOriginal1 = [1, 2, 3, 4, 5];

let arrayIntegers1 = arrayIntegersOriginal1.splice(0,2); // returns [1, 2]; original array: [3, 4, 5]

let arrayIntegers2 = arrayIntegersOriginal2.splice(3); // returns [4, 5]; original array: [1, 2, 3]

let arrayIntegers3 = arrayIntegersOriginal3.splice(3, 1, "a", "b", "c"); //returns [4]; original array: [1, 2, 3, "a", "b", "c", 5]

**Note:** Splice method modifies the original array and returns the deleted array.

1. **What is the difference between slice and splice**

Some of the major difference in a tabular form

| **Slice** | **Splice** |
| --- | --- |
| Doesn't modify the original array(immutable) | Modifies the original array(mutable) |
| Returns the subset of original array | Returns the deleted elements as array |
| Used to pick the elements from array | Used to insert or delete elements to/from array |

1. **What is the difference between == and === operators**

== compare the both value

=== compare the vale with data types

Some of the examples which covers the above cases,

0 == false // true

0 === false // false

1 == "1" // true

1 === "1" // false

null == undefined // true

null === undefined // false

'0' == false // true

'0' === false // false

[]==[] or []===[] //false, refer different objects in memory

{}=={} or {}==={} //false, refer different objects in memory

1. **What are lambda or arrow functions**

An arrow function is a shorter syntax for a function expression and does not have its own **this, arguments, super, or new.target**. These functions are best suited for non-method functions, and they cannot be used as constructors.

1. **What is a first class function**

In Javascript, functions are first class objects. First-class functions means when functions in that language are treated like any other variable.

For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable. For example, in the below example, handler functions assigned to a listener

const handler = () => console.log ('This is a click handler function');

document.addEventListener ('click', handler);

1. **What is a first order function**

First-order function is a function that doesn’t accept another function as an argument and doesn’t return a function as its return value.

const firstOrder = () => console.log ('I am a first order function!');

1. **What is a higher order function**

Higher-order function is a function that accepts another function as an argument or returns a function as a return value or both.

const firstOrderFunc = () => console.log ('Hello, I am a First order function');

const higherOrder = ReturnFirstOrderFunc => ReturnFirstOrderFunc();

higherOrder(firstOrderFunc);

1. **What is a unary function**

Unary function (i.e. monadic) is a function that accepts exactly one argument. It stands for a single argument accepted by a function.

Let us take an example of unary function,

const unaryFunction = a => console.log (a + 10); // Add 10 to the given argument and display the value

1. **What is the currying function**

Currying is the process of taking a function with multiple arguments and turning it into a sequence of functions each with only a single argument. Currying is named after a mathematician **Haskell Curry**. By applying currying, a n-ary function turns it into a unary function.

Let's take an example of n-ary function and how it turns into a currying function,

const multiArgFunction = (a, b, c) => a + b + c;

console.log(multiArgFunction(1,2,3));// 6

const curryUnaryFunction = a => b => c => a + b + c;

curryUnaryFunction (1); // returns a function: b => c => 1 + b + c

curryUnaryFunction (1) (2); // returns a function: c => 3 + c

curryUnaryFunction (1) (2) (3); // returns the number 6

Curried functions are great to improve **code reusability** and **functional composition**.

1. **What is a pure function**

A **Pure function** is a function where the return value is only determined by its arguments without any side effects. i.e, If you call a function with the same arguments 'n' number of times and 'n' number of places in the application then it will always return the same value.

Let's take an example to see the difference between pure and impure functions,

//Impure

let numberArray = [];

const impureAddNumber = number => numberArray.push(number);

//Pure

const pureAddNumber = number => argNumberArray =>

argNumberArray.concat([number]);

//Display the results

console.log (impureAddNumber(6)); // returns 1

console.log (numberArray); // returns [6]

console.log (pureAddNumber(7) (numberArray)); // returns [6, 7]

console.log (numberArray); // returns [6]

1. **What is the purpose of the let keyword**

The let statement declares a **block scope local variable**. Hence the variables defined with let keyword are limited in scope to the block, statement, or expression on which it is used. Whereas variables declared with the var keyword used to define a variable globally, or locally to an entire function regardless of block scope.

Let's take an example to demonstrate the usage,

let counter = 30;

if (counter === 30) {

let counter = 31;

console.log(counter); // 31

}

console.log(counter); // 30 (because the variable in if block won't exist here)

1. **What is the difference between let and var**

You can list out the differences in a tabular format

| **var** | **let** |
| --- | --- |
| It is been available from the beginning of JavaScript | Introduced as part of ES6 |
| It has function scope | It has block scope |
| Variables will be hoisted | Hoisted but not initialized |

Let's take an example to see the difference,

function userDetails(username) {

if(username) {

console.log(salary); // undefined due to hoisting

console.log(age); // ReferenceError: Cannot access 'age' before initialization

let age = 30;

var salary = 10000;

}

console.log(salary); //10000 (accessible to due function scope)

console.log(age); //error: age is not defined(due to block scope)

}

userDetails('John');

1. **What is the reason to choose the name let as a keyword**

let is a mathematical statement that was adopted by early programming languages like **Scheme** and **Basic**. It has been borrowed from dozens of other languages that use let already as a traditional keyword as close to var as possible.

1. **What is the Temporal Dead Zone**

The Temporal Dead Zone is a behavior in JavaScript that occurs when declaring a variable with the let and const keywords, but not with var. In ECMAScript 6, accessing a let or const variable before its declaration (within its scope) causes a ReferenceError. The time span when that happens, between the creation of a variable’s binding and its declaration, is called the temporal dead zone.

Let's see this behavior with an example,

function somemethod() {

console.log(counter1); // undefined

console.log(counter2); // ReferenceError

var counter1 = 1;

let counter2 = 2;

}

1. **What is IIFE(Immediately Invoked Function Expression)**

IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined. The signature of it would be as below,

(function ()

{

// logic here

}

)

();

The primary reason to use an IIFE is to obtain data privacy because any variables declared within the IIFE cannot be accessed by the outside world. i.e, If you try to access variables with IIFE then it throws an error as below,

(function ()

{

var message = "IIFE";

console.log(message);

}

)

();

console.log(message); //Error: message is not defined

1. **How do you decode or encode a URL in JavaScript?**

encodeURI() function is used to encode an URL. This function requires a URL string as a parameter and return that encoded string. decodeURI() function is used to deocde an URL. This function requires an encoded URL string as parameter and return that decoded string.

**Note:** If you want to encode characters such as / ? : @ & = + $ # then you need to use encodeURIComponent().

let uri = "employeeDetails?name=john&occupation=manager";

let encoded\_uri = encodeURI(uri);

let decoded\_uri = decodeURI(encoded\_uri);

1. **What is memoization**

Memoization is a programming technique which attempts to increase a function’s performance by caching its previously computed results. Each time a memoized function is called, its parameters are used to index the cache. If the data is present, then it can be returned, without executing the entire function. Otherwise the function is executed and then the result is added to the cache. Let's take an example of adding function with memoization,

const memoizAddition = () => {

let cache = {};

return (value) => {

if (value in cache) {

console.log('Fetching from cache');

return cache[value]; // Here, cache.value cannot be used as property name starts with the number which is not a valid JavaScript identifier. Hence, can only be accessed using the square bracket notation.

}

else {

console.log('Calculating result');

let result = value + 20;

cache[value] = result;

return result;

}

}

}

// returned function from memoizAddition

const addition = memoizAddition();

console.log(addition(20)); //output: 40 calculated

console.log(addition(20)); //output: 40 cached

1. **What is Hoisting**

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution. Remember that JavaScript only hoists declarations, not initialisation. Let's take a simple example of variable hoisting,

console.log(message); //output : undefined

var message = 'The variable Has been hoisted';

The above code looks like as below to the interpreter,

var message;

console.log(message);

message = 'The variable Has been hoisted';

1. **What are classes in ES6**

In ES6, Javascript classes are primarily syntactic sugar over JavaScript’s existing prototype-based inheritance. For example, the prototype based inheritance written in function expression as below,

function Bike(model,color) {

this.model = model;

this.color = color;

}

Bike.prototype.getDetails = function() {

return this.model + ' bike has' + this.color + ' color';

};

Whereas ES6 classes can be defined as an alternative

class Bike{

constructor(color, model) {

this.color= color;

this.model= model;

}

getDetails() {

return this.model + ' bike has' + this.color + ' color';

}

}

1. **What are closures**

A closure is the combination of a function and the lexical environment within which that function was declared. i.e, It is an inner function that has access to the outer or enclosing function’s variables. The closure has three scope chains

* 1. Own scope where variables defined between its curly brackets
  2. Outer function’s variables
  3. Global variables

Let's take an example of closure concept,

function Welcome(name){

var greetingInfo = function(message){

console.log(message+' '+name);

}

return greetingInfo;

}

var myFunction = Welcome('John');

myFunction('Welcome '); //Output: Welcome John

myFunction('Hello Mr.'); //output: Hello Mr.John

1. **What is scope in javascript**

Scope is the accessibility of variables, functions, and objects in some particular part of your code during runtime. In other words, scope determines the visibility of variables and other resources in areas of your code.

1. **What is a service worker**

A Service worker is basically a script (JavaScript file) that runs in the background, separate from a web page and provides features that don't need a web page or user interaction. Some of the major features of service workers are Rich offline experiences(offline first web application development), periodic background syncs, push notifications, intercept and handle network requests and programmatically managing a cache of responses.

1. **How do you manipulate DOM using a service worker**

Service worker can't access the DOM directly. But it can communicate with the pages it controls by responding to messages sent via the postMessage interface, and those pages can manipulate the DOM.

1. **What is IndexedDB**

IndexedDB is a low-level API for client-side storage of larger amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data.

1. **What is web storage**

Web storage is an API that provides a mechanism by which browsers can store key/value pairs locally within the user's browser, in a much more intuitive fashion than using cookies. The web storage provides two mechanisms for storing data on the client.

* 1. **Local storage:** It stores data for current origin with no expiration date.
  2. **Session storage:** It stores data for one session and the data is lost when the browser tab is closed.

1. **What is a Cookie**

A cookie is a piece of data that is stored on your computer to be accessed by your browser. Cookies are saved as key/value pairs. For example, you can create a cookie named username as below,

document.cookie = "username=John";

1. **Why do you need a Cookie**

Cookies are used to remember information about the user profile(such as username). It basically involves two steps,

* 1. When a user visits a web page, the user profile can be stored in a cookie.
  2. Next time the user visits the page, the cookie remembers the user profile.

1. **What are the options in a cookie**

There are few below options available for a cookie,

* 1. By default, the cookie is deleted when the browser is closed but you can change this behavior by setting expiry date (in UTC time).

document.cookie = "username=John; expires=Sat, 8 Jun 2019 12:00:00 UTC";

* 1. By default, the cookie belongs to a current page. But you can tell the browser what path the cookie belongs to using a path parameter.

document.cookie = "username=John; path=/services";

1. **How do you delete a cookie**

You can delete a cookie by setting the expiry date as a passed date. You don't need to specify a cookie value in this case. For example, you can delete a username cookie in the current page as below.

document.cookie = "username=; expires=Fri, 07 Jun 2019 00:00:00 UTC; path=/;";

**Note:** You should define the cookie path option to ensure that you delete the right cookie. Some browsers doesn't allow to delete a cookie unless you specify a path parameter.

1. **What are the differences between cookie, local storage and session storage**

Below are some of the differences between cookie, local storage and session storage,

| **Feature** | **Cookie** | **Local storage** | **Session storage** |
| --- | --- | --- | --- |
| Accessed on client or server side | Both server-side & client-side | client-side only | client-side only |
| Lifetime | As configured using Expires option | until deleted | until tab is closed |
| SSL support | Supported | Not supported | Not supported |
| Maximum data size | 4KB | 5 MB | 5MB |

1. **What is the main difference between localStorage and sessionStorage**

LocalStorage is the same as SessionStorage but it persists the data even when the browser is closed and reopened(i.e it has no expiration time) whereas in sessionStorage data gets cleared when the page session ends.

1. **How do you access web storage**

The Window object implements the WindowLocalStorage and WindowSessionStorage objects which has localStorage(window.localStorage) and sessionStorage(window.sessionStorage) properties respectively. These properties create an instance of the Storage object, through which data items can be set, retrieved and removed for a specific domain and storage type (session or local). For example, you can read and write on local storage objects as below

localStorage.setItem('logo', document.getElementById('logo').value);

localStorage.getItem('logo');

1. **What are the methods available on session storage**

The session storage provided methods for reading, writing and clearing the session data

// Save data to sessionStorage

sessionStorage.setItem('key', 'value');

// Get saved data from sessionStorage

let data = sessionStorage.getItem('key');

// Remove saved data from sessionStorage

sessionStorage.removeItem('key');

// Remove all saved data from sessionStorage

sessionStorage.clear();

1. **What is a storage event and its event handler**

The StorageEvent is an event that fires when a storage area has been changed in the context of another document. Whereas onstorage property is an EventHandler for processing storage events. The syntax would be as below

window.onstorage = functionRef;

Let's take the example usage of onstorage event handler which logs the storage key and it's values

window.onstorage = function(e) {

console.log('The ' + e.key +

' key has been changed from ' + e.oldValue +

' to ' + e.newValue + '.');

};

1. **Why do you need web storage**

Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance. Also, the information is never transferred to the server. Hence this is a more recommended approach than Cookies.

1. **What are the restrictions of web workers on DOM**

Web Workers don't have access to below JavaScript objects since they are defined in an external file

* 1. Window object
  2. Document object
  3. Parent object

1. **What is a promise**

A promise is an object that may produce a single value some time in the future with either a resolved value or a reason that it’s not resolved(for example, network error). It will be in one of the 3 possible states: fulfilled, rejected, or pending.

The syntax of Promise creation looks like below,

const promise = new Promise(function(resolve, reject) {

// promise description

})

The usage of a promise would be as below,

const promise = new Promise(resolve => {

setTimeout(() => {

resolve("I'm a Promise!");

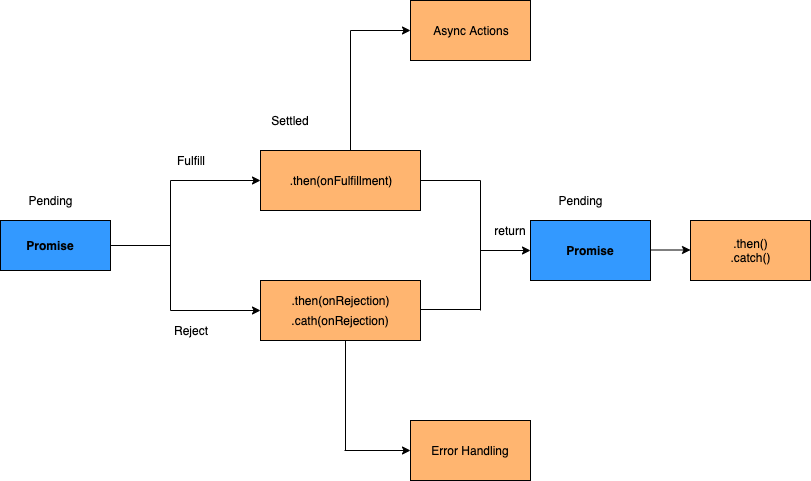
}, 5000);

}, reject => {

});

promise.then(value => console.log(value));

The action flow of a promise will be as below,

[](https://github.com/sudheerj/javascript-interview-questions/blob/master/images/promises.png)

1. **Why do you need a promise**

Promises are used to handle asynchronous operations. They provide an alternative approach for callbacks by reducing the callback hell and writing the cleaner code.

1. **What are the three states of promise**

Promises have three states:

* 1. **Pending:** This is an initial state of the Promise before an operation begins
  2. **Fulfilled:** This state indicates that the specified operation was completed.
  3. **Rejected:** This state indicates that the operation did not complete. In this case an error value will be thrown.

1. **What is a callback function**

A callback function is a function passed into another function as an argument. This function is invoked inside the outer function to complete an action. Let's take a simple example of how to use callback function

function callbackFunction(name) {

console.log('Hello ' + name);

}

function outerFunction(callback) {

let name = prompt('Please enter your name.');

callback(name);

}

outerFunction(callbackFunction);

1. **Why do we need callbacks**

The callbacks are needed because javascript is an event driven language. That means instead of waiting for a response javascript will keep executing while listening for other events. Let's take an example with the first function invoking an API call(simulated by setTimeout) and the next function which logs the message.

function firstFunction(){

// Simulate a code delay

setTimeout( function(){

console.log('First function called');

}, 1000 );

}

function secondFunction(){

console.log('Second function called');

}

firstFunction();

secondFunction();

Output

// Second function called

// First function called

As observed from the output, javascript didn't wait for the response of the first function and the remaining code block got executed. So callbacks are used in a way to make sure that certain code doesn’t execute until the other code finishes execution.

1. **What is a callback hell**

Callback Hell is an anti-pattern with multiple nested callbacks which makes code hard to read and debug when dealing with asynchronous logic. The callback hell looks like below,

async1(function(){

async2(function(){

async3(function(){

async4(function(){

....

});

});

});

});

1. **What is promise chaining**

The process of executing a sequence of asynchronous tasks one after another using promises is known as Promise chaining. Let's take an example of promise chaining for calculating the final result,

new Promise(function(resolve, reject) {

setTimeout(() => resolve(1), 1000);

}).then(function(result) {

console.log(result); // 1

return result \* 2;

}).then(function(result) {

console.log(result); // 2

return result \* 3;

}).then(function(result) {

console.log(result); // 6

return result \* 4;

});

In the above handlers, the result is passed to the chain of .then() handlers with the below work flow,

* 1. The initial promise resolves in 1 second,
  2. After that .then handler is called by logging the result(1) and then return a promise with the value of result \* 2.
  3. After that the value passed to the next .then handler by logging the result(2) and return a promise with result \* 3.
  4. Finally the value passed to the last .then handler by logging the result(6) and return a promise with result \* 4.

1. **What is promise.all**

Promise.all is a promise that takes an array of promises as an input (an iterable), and it gets resolved when all the promises get resolved or any one of them gets rejected. For example, the syntax of promise.all method is below,

Promise.all([Promise1, Promise2, Promise3]) .then(result) => { console.log(result) }) .catch(error => console.log(`Error in promises ${error}`))

**Note:** Remember that the order of the promises(output the result) is maintained as per input order.

1. **What is the purpose of the race method in promise**

Promise.race() method will return the promise instance which is firstly resolved or rejected. Let's take an example of race() method where promise2 is resolved first

var promise1 = new Promise(function(resolve, reject) {

setTimeout(resolve, 500, 'one');

});

var promise2 = new Promise(function(resolve, reject) {

setTimeout(resolve, 100, 'two');

});

Promise.race([promise1, promise2]).then(function(value) {

console.log(value); // "two" // Both promises will resolve, but promise2 is faster

});

1. **What is a strict mode in javascript**

Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a “strict” operating context. This way it prevents certain actions from being taken and throws more exceptions. The literal expression "use strict"; instructs the browser to use the javascript code in the Strict mode.

1. **Why do you need strict mode**

Strict mode is useful to write "secure" JavaScript by notifying "bad syntax" into real errors. For example, it eliminates accidentally creating a global variable by throwing an error and also throws an error for assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object.

1. **How do you declare strict mode**

The strict mode is declared by adding "use strict"; to the beginning of a script or a function. If declared at the beginning of a script, it has global scope.

"use strict";

x = 3.14; // This will cause an error because x is not declared

and if you declare inside a function, it has local scope

x = 3.14; // This will not cause an error.

myFunction();

function myFunction() {

"use strict";

y = 3.14; // This will cause an error

}

1. **What is the purpose of the delete operator**

The delete keyword is used to delete the property as well as its value.

var user= {name: "John", age:20};

delete user.age;

console.log(user); // {name: "John"}

1. **What is the typeof operator**

You can use the JavaScript typeof operator to find the type of a JavaScript variable. It returns the type of a variable or an expression.

typeof "John Abraham" // Returns "string"

typeof (1 + 2) // Returns "number"

1. **What is undefined property**

The undefined property indicates that a variable has not been assigned a value, or not declared at all. The type of undefined value is undefined too.

var user; // Value is undefined, type is undefined

console.log(typeof(user)) //undefined

Any variable can be emptied by setting the value to undefined.

user = undefined

1. **What is null value**

The value null represents the intentional absence of any object value. It is one of JavaScript's primitive values. The type of null value is object. You can empty the variable by setting the value to null.

var user = null;

console.log(typeof(user)) //object

1. **What is the difference between null and undefined**

Below are the main differences between null and undefined,

| **Null** | **Undefined** |
| --- | --- |
| It is an assignment value which indicates that variable points to no object. | It is not an assignment value where a variable has been declared but has not yet been assigned a value. |
| Type of null is object | Type of undefined is undefined |
| The null value is a primitive value that represents the null, empty, or non-existent reference. | The undefined value is a primitive value used when a variable has not been assigned a value. |
| Indicates the absence of a value for a variable | Indicates absence of variable itself |
| Converted to zero (0) while performing primitive operations | Converted to NaN while performing primitive operations |

1. **What is eval**

The eval() function evaluates JavaScript code represented as a string. The string can be a JavaScript expression, variable, statement, or sequence of statements.

console.log(eval('1 + 2')); // 3

1. **What is the difference between window and document**

Below are the main differences between window and document,

| **Window** | **Document** |
| --- | --- |
| It is the root level element in any web page | It is the direct child of the window object. This is also known as Document Object Model(DOM) |
| By default window object is available implicitly in the page | You can access it via window.document or document. |
| It has methods like alert(), confirm() and properties like document, location | It provides methods like getElementById, getElementsByTagName, createElement etc |

1. **How do you access history in javascript**

The window.history object contains the browser's history. You can load previous and next URLs in the history using back() and next() methods.

function goBack() {

window.history.back()

}

function goForward() {

window.history.forward()

}

**Note:** You can also access history without window prefix.

1. **What are the differences between undeclared and undefined variables**

Below are the major differences between undeclared and undefined variables,

| **undeclared** | **Undefined** |
| --- | --- |
| These variables do not exist in a program and are not declared | These variables declared in the program but have not assigned any value |
| If you try to read the value of an undeclared variable, then a runtime error is encountered | If you try to read the value of an undefined variable, an undefined value is returned. |

1. **What are global variables**

Global variables are those that are available throughout the length of the code without any scope. The var keyword is used to declare a local variable but if you omit it then it will become global variable

msg = "Hello" // var is missing, it becomes global variable

1. **What is NaN property**

The NaN property is a global property that represents "Not-a-Number" value. i.e, It indicates that a value is not a legal number. It is very rare to use NaN in a program but it can be used as return value for few cases

Math.sqrt(-1)

parseInt("Hello")

1. **What is an event flow**

Event flow is the order in which event is received on the web page. When you click an element that is nested in various other elements, before your click actually reaches its destination, or target element, it must trigger the click event for each of its parent elements first, starting at the top with the global window object. There are two ways of event flow

* 1. Top to Bottom(Event Capturing)
  2. Bottom to Top (Event Bubbling)

1. **What is event bubbling**

Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same nesting hierarchy till it reaches the outermost DOM element.

1. **What is event capturing**

Event capturing is a type of event propagation where the event is first captured by the outermost element, and then successively triggers on the descendants (children) of the target element in the same nesting hierarchy till it reaches the innermost DOM element.

1. **How do you submit a form using JavaScript**

You can submit a form using document.forms[0].submit(). All the form input's information is submitted using onsubmit event handler

function submit() {

document.forms[0].submit();}

1. **What are the pros and cons of promises over callbacks**

Below are the list of pros and cons of promises over callbacks,

**Pros:**

* 1. It avoids callback hell which is unreadable
  2. Easy to write sequential asynchronous code with .then()
  3. Easy to write parallel asynchronous code with Promise.all()
  4. Solves some of the common problems of callbacks(call the callback too late, too early, many times and swallow errors/exceptions)

**Cons:**

* 1. It makes little complex code
  2. You need to load a polyfill if ES6 is not supported

1. **What is the use of preventDefault method**

The preventDefault() method cancels the event if it is cancelable, meaning that the default action or behaviour that belongs to the event will not occur. For example, prevent form submission when clicking on submit button and prevent opening the page URL when clicking on hyperlink are some common use cases.

document.getElementById("link").addEventListener("click", function(event){

event.preventDefault();

});

**Note:** Remember that not all events are cancelable.

1. **What is the use of stopPropagation method**

The stopPropagation method is used to stop the event from bubbling up the event chain. For example, the below nested divs with stopPropagation method prevents default event propagation when clicking on nested div(Div1)

<p>Click DIV1 Element</p>

<div onclick="secondFunc()">DIV 2

<div onclick="firstFunc(event)">DIV 1</div>

</div>

<script>

function firstFunc(event) {

alert("DIV 1");

event.stopPropagation();

}

function secondFunc() {

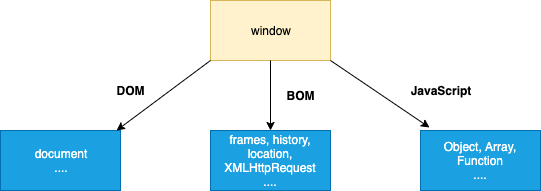
alert("DIV 2");

}

</script>

1. **What is BOM**

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser. It consists of the objects navigator, history, screen, location and document which are children of the window. The Browser Object Model is not standardized and can change based on different browsers.

[](https://github.com/sudheerj/javascript-interview-questions/blob/master/images/bom.png)

1. **What is the use of setTimeout**

The setTimeout() method is used to call a function or evaluate an expression after a specified number of milliseconds. For example, let's log a message after 2 seconds using setTimeout method,

setTimeout(function(){ console.log("Good morning"); }, 2000);

1. **What is the use of setInterval**

The setInterval() method is used to call a function or evaluate an expression at specified intervals (in milliseconds). For example, let's log a message after 2 seconds using setInterval method,

setInterval(function(){ console.log("Good morning"); }, 2000);

1. **What is an event delegation**

Event delegation is a technique for listening to events where you delegate a parent element as the listener for all of the events that happen inside it.

For example, if you wanted to detect field changes in inside a specific form, you can use event delegation technique,

var form = document.querySelector('#registration-form');

// Listen for changes to fields inside the form

form.addEventListener('input', function (event) {

// Log the field that was changed

console.log(event.target);

}, false);

1. **What is ECMAScript**

ECMAScript is the scripting language that forms the basis of JavaScript. ECMAScript standardized by the ECMA International standards organization in the ECMA-262 and ECMA-402 specifications. The first edition of ECMAScript was released in 1997.

1. **What is JSON**

JSON (JavaScript Object Notation) is a lightweight format that is used for data interchanging. It is based on a subset of JavaScript language in the way objects are built in JavaScript.

1. **What is the purpose JSON stringify**

When sending data to a web server, the data has to be in a string format. You can achieve this by converting JSON object into a string using stringify() method.

var userJSON = {'name': 'John', age: 31}

var userString = JSON.stringify(user);

console.log(userString); //"{"name":"John","age":31}"

1. **How do you parse JSON string**

When receiving the data from a web server, the data is always in a string format. But you can convert this string value to a javascript object using parse() method.

var userString = '{"name":"John","age":31}';

var userJSON = JSON.parse(userString);

console.log(userJSON);// {name: "John", age: 31}

1. **Why do you need JSON**

When exchanging data between a browser and a server, the data can only be text. Since JSON is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

1. **What are PWAs**

Progressive web applications (PWAs) are a type of mobile app delivered through the web, built using common web technologies including HTML, CSS and JavaScript. These PWAs are deployed to servers, accessible through URLs, and indexed by search engines.

1. **What is the purpose of clearTimeout method**

The clearTimeout() function is used in javascript to clear the timeout which has been set by setTimeout()function before that. i.e, The return value of setTimeout() function is stored in a variable and it’s passed into the clearTimeout() function to clear the timer.

For example, the below setTimeout method is used to display the message after 3 seconds. This timeout can be cleared by the clearTimeout() method.

<script>

var msg;

function greeting() {

alert('Good morning');

}

function start() {

msg =setTimeout(greeting, 3000);

}

function stop() {

clearTimeout(msg);

}

</script>

1. **What is the purpose of clearInterval method**

The clearInterval() function is used in javascript to clear the interval which has been set by setInterval() function. i.e, The return value returned by setInterval() function is stored in a variable and it’s passed into the clearInterval() function to clear the interval.

For example, the below setInterval method is used to display the message for every 3 seconds. This interval can be cleared by the clearInterval() method.

<script>

var msg;

function greeting() {

alert('Good morning');

}

function start() {

msg = setInterval(greeting, 3000);

}

function stop() {

clearInterval(msg);

}

</script>

1. **How do you check if a key exists in an object**
   1. **Using in operator:** You can use the in operator whether a key exists in an object or not

"key" in obj

and If you want to check if a key doesn't exist, remember to use parenthesis,

!("key" in obj)

* 1. **Using hasOwnProperty method:** You can use hasOwnProperty to particularly test for properties of the object instance (and not inherited properties)

obj.hasOwnProperty("key") // true

1. **How do you loop through or enumerate javascript object**

You can use the for-in loop to loop through javascript object. You can also make sure that the key you get is an actual property of an object, and doesn't come from the prototype using hasOwnProperty method.

var object = {

"k1": "value1",

"k2": "value2",

"k3": "value3"

};

for (var key in object) {

if (object.hasOwnProperty(key)) {

console.log(key + " -> " + object[key]); // k1 -> value1 ...

}

}

1. **How do you test for an empty object**

There are different solutions based on ECMAScript versions

* 1. **Using Object entries(ECMA 7+):** You can use object entries length along with constructor type.

Object.entries(obj).length === 0 && obj.constructor === Object // Since date object length is 0, you need to check constructor check as well

* 1. **Using Object keys(ECMA 5+):** You can use object keys length along with constructor type.

Object.keys(obj).length === 0 && obj.constructor === Object // Since date object length is 0, you need to check constructor check as well

* 1. **Using for-in with hasOwnProperty(Pre-ECMA 5):** You can use a for-in loop along with hasOwnProperty.

function isEmpty(obj) {

for(var prop in obj) {

if(obj.hasOwnProperty(prop)) {

return false;

}

}

return JSON.stringify(obj) === JSON.stringify({});

}

1. **How do you display the current date in javascript**

You can use new Date() to generate a new Date object containing the current date and time. For example, let's display the current date in mm/dd/yyyy

var today = new Date();

var dd = String(today.getDate()).padStart(2, '0');

var mm = String(today.getMonth() + 1).padStart(2, '0'); //January is 0!

var yyyy = today.getFullYear();

today = mm + '/' + dd + '/' + yyyy;

document.write(today);

1. **How do you compare two date objects**

You need to use date.getTime() method to compare date values instead of comparison operators (==, !=, ===, and !== operators)

var d1 = new Date();

var d2 = new Date(d1);

console.log(d1.getTime() === d2.getTime()); //True

console.log(d1 === d2); // False

1. **How do you trim a string in javascript**

JavaScript provided a trim method on string types to trim any whitespaces present at the beginning or ending of the string.

" Hello World ".trim(); //Hello World

If your browser(<IE9) doesn't support this method then you can use below polyfill.

if (!String.prototype.trim) {

(function() {

// Make sure we trim BOM and NBSP

var rtrim = /^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g;

String.prototype.trim = function() {

return this.replace(rtrim, '');

};

})();

}

1. **How do you add a key value pair in javascript**

There are two possible solutions to add new properties to an object. Let's take a simple object to explain these solutions.

var object = {

key1: value1,

key2: value2

};

* 1. **Using dot notation:** This solution is useful when you know the name of the property

object.key3 = "value3";

* 1. **Using square bracket notation:** This solution is useful when the name of the property is dynamically determined.

obj["key3"] = "value3";

1. **What is the way to find the number of parameters expected by a function**

You can use function.length syntax to find the number of parameters expected by a function. Let's take an example of sum function to calculate the sum of numbers,

function sum(num1, num2, num3, num4){

return num1 + num2 + num3 + num4;

}

sum.length // 4 is the number of parameters expected.

1. **What is a polyfill**

A polyfill is a piece of JS code used to provide modern functionality on older browsers that do not natively support it. For example, Silverlight plugin polyfill can be used to mimic the functionality of an HTML Canvas element on Microsoft Internet Explorer 7.

1. **What is tree shaking**

Tree shaking is a form of dead code elimination. It means that unused modules will not be included in the bundle during the build process and for that it relies on the static structure of ES2015 module syntax,( i.e. import and export). Initially this has been popularized by the ES2015 module bundler rollup.

1. **What is the need of tree shaking**

Tree Shaking can significantly reduce the code size in any application. i.e, The less code we send over the wire the more performant the application will be. For example, if we just want to create a “Hello World” Application using SPA frameworks then it will take around a few MBs, but by tree shaking it can bring down the size to just a few hundred KBs. Tree shaking is implemented in Rollup and Webpack bundlers.

1. **Is it recommended to use eval**

No, it allows arbitrary code to be run which causes a security problem. As we know that the eval() function is used to run text as code. In most of the cases, it should not be necessary to use it.

1. **What would be the result of 1+2+'3'**

The output is going to be 33. Since 1 and 2 are numeric values, the result of the first two digits is going to be a numeric value 3. The next digit is a string type value because of that the addition of numeric value 3 and string type value 3 is just going to be a concatenation value 33.

1. **How do you convert date to another timezone in javascript**

You can use the toLocaleString() method to convert dates in one timezone to another. For example, let's convert current date to British English timezone as below,

console.log(event.toLocaleString('en-GB', { timeZone: 'UTC' })); //29/06/2019, 09:56:00

1. **What is a conditional operator in javascript**

The conditional (ternary) operator is the only JavaScript operator that takes three operands which acts as a shortcut for if statements.

var isAuthenticated = false;

console.log(isAuthenticated ? 'Hello, welcome' : 'Sorry, you are not authenticated'); //Sorry, you are not authenticated

1. **What are the ways to execute javascript after page load**

You can execute javascript after page load in many different ways,

* 1. **window.onload:**

window.onload = function ...

* 1. **document.onload:**

document.onload = function ...

* 1. **body onload:**

<body onload="script();">

1. **What is the difference between proto and prototype**

The \_\_proto\_\_ object is the actual object that is used in the lookup chain to resolve methods, etc. Whereas prototype is the object that is used to build \_\_proto\_\_ when you create an object with new

( new Employee ).\_\_proto\_\_ === Employee.prototype;

( new Employee ).prototype === undefined;

1. **What is a freeze method**

The **freeze()** method is used to freeze an object. Freezing an object does not allow adding new properties to an object,prevents from removing and prevents changing the enumerability, configurability, or writability of existing properties. i.e, It returns the passed object and does not create a frozen copy.

const obj = {

prop: 100

};

Object.freeze(obj);

obj.prop = 200; // Throws an error in strict mode

console.log(obj.prop); //100

**Note:** It causes a TypeError if the argument passed is not an object.

1. **What is the purpose of freeze method**

Below are the main benefits of using freeze method,

* 1. It is used for freezing objects and arrays.
  2. It is used to make an object immutable.

1. **Why do I need to use freeze method**

In the Object-oriented paradigm, an existing API contains certain elements that are not intended to be extended, modified, or re-used outside of their current context. Hence it works as the final keyword which is used in various languages.

1. **What is a rest parameter**

Rest parameter is an improved way to handle function parameters which allows us to represent an indefinite number of arguments as an array. The syntax would be as below,

function f(a, b, ...theArgs) {

// ...

}

For example, let's take a sum example to calculate on dynamic number of parameters,

function total(…args){

let sum = 0;

for(let i of args){

sum+=i;

}

return sum;

}

console.log(fun(1,2)); //3

console.log(fun(1,2,3)); //6

console.log(fun(1,2,3,4)); //13

console.log(fun(1,2,3,4,5)); //15

**Note:** Rest parameter is added in ES2015 or ES6

1. **What is a spread operator**

Spread operator allows iterables( arrays / objects / strings ) to be expanded into single arguments/elements. Let's take an example to see this behavior,

function calculateSum(x, y, z) {

return x + y + z;

}

const numbers = [1, 2, 3];

console.log(calculateSum(...numbers)); // 6

1. **What is the purpose of seal method**

The **Object.seal()** method is used to seal an object, by preventing new properties from being added to it and marking all existing properties as non-configurable. But values of present properties can still be changed as long as they are writable. Let's see the below example to understand more about seal() method

const object = {

property: 'Welcome JS world'

};

Object.seal(object);

object.property = 'Welcome to object world';

console.log(Object.isSealed(object)); // true

delete object.property; // You cannot delete when sealed

console.log(object.property); //Welcome to object world

1. **What are the applications of seal method**

Below are the main applications of Object.seal() method,

* 1. It is used for sealing objects and arrays.
  2. It is used to make an object immutable.

1. **What are the differences between freeze and seal methods**

If an object is frozen using the Object.freeze() method then its properties become immutable and no changes can be made in them whereas if an object is sealed using the Object.seal() method then the changes can be made in the existing properties of the object

1. **What is the main difference between Object.values and Object.entries method**

The Object.values() method's behavior is similar to Object.entries() method but it returns an array of values instead [key,value] pairs.

const object = {

a: 'Good morning',

b: 100

};

for (let value of Object.values(object)) {

console.log(`${value}`); // 'Good morning'

100

}

1. **How can you get the list of keys of any object**

You can use the Object.keys() method which is used to return an array of a given object's own property names, in the same order as we get with a normal loop. For example, you can get the keys of a user object,

const user = {

name: 'John',

gender: 'male',

age: 40

};

console.log(Object.keys(user)); //['name', 'gender', 'age']

1. **What is a WeakSet**

WeakSet is used to store a collection of weakly(weak references) held objects. The syntax would be as follows,

new WeakSet([iterable]);

Let's see the below example to explain it's behavior,

var ws = new WeakSet();

var user = {};

ws.add(user);

ws.has(user); // true

ws.delete(user); // removes user from the set

ws.has(user); // false, user has been removed

1. **What are the differences between WeakSet and Set**

The main difference is that references to objects in Set are strong while references to objects in WeakSet are weak. i.e, An object in WeakSet can be garbage collected if there is no other reference to it. Other differences are,

* 1. Sets can store any value Whereas WeakSets can store only collections of objects
  2. WeakSet does not have size property unlike Set
  3. WeakSet does not have methods such as clear, keys, values, entries, forEach.
  4. WeakSet is not iterable.

1. **What is a WeakMap**

The WeakMap object is a collection of key/value pairs in which the keys are weakly referenced. In this case, keys must be objects and the values can be arbitrary values. The syntax is looking like as below,

new WeakMap([iterable])

Let's see the below example to explain it's behavior,

var ws = new WeakMap();

var user = {};

ws.set(user);

ws.has(user); // true

ws.delete(user); // removes user from the map

ws.has(user); // false, user has been removed

1. **What are the differences between WeakMap and Map**

The main difference is that references to key objects in Map are strong while references to key objects in WeakMap are weak. i.e, A key object in WeakMap can be garbage collected if there is no other reference to it. Other differences are,

* 1. Maps can store any key type Whereas WeakMaps can store only collections of key objects
  2. WeakMap does not have size property unlike Map
  3. WeakMap does not have methods such as clear, keys, values, entries, forEach.
  4. WeakMap is not iterable.

1. **How do you encode an URL**

The encodeURI() function is used to encode complete URI which has special characters except (, / ? : @ & = + $ #) characters.

var uri = 'https://mozilla.org/?x=шеллы';

var encoded = encodeURI(uri);

console.log(encoded); // https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B

1. **How do you decode an URL**

The decodeURI() function is used to decode a Uniform Resource Identifier (URI) previously created by encodeURI().

var uri = 'https://mozilla.org/?x=шеллы';

var encoded = encodeURI(uri);

console.log(encoded); // https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B

try {

console.log(decodeURI(encoded)); // "https://mozilla.org/?x=шеллы"

} catch(e) { // catches a malformed URI

console.error(e);

}

1. **What is an anonymous function**

An anonymous function is a function without a name! Anonymous functions are commonly assigned to a variable name or used as a callback function. The syntax would be as below,

const myFunction = function(){ //Anonymous function assigned to a variable

//do something

};

1. **What are the advantages of Getters and Setters**

Below are the list of benefits of Getters and Setters,

* 1. They provide simpler syntax
  2. They are used for defining computed properties, or accessors in JS.
  3. Useful to provide equivalence relation between properties and methods
  4. They can provide better data quality
  5. Useful for doing things behind the scenes with the encapsulated logic.

1. **What are primitive data types**

A primitive data type is data that has a primitive value (which has no properties or methods). There are 7 types of primitive data types.

* 1. string
  2. number
  3. boolean
  4. null
  5. undefined
  6. bigint
  7. symbol

1. **What are the different error names from error object**

There are 6 different types of error names returned from error object,

| **Error Name** | **Description** |
| --- | --- |
| EvalError | An error has occurred in the eval() function |
| RangeError | An error has occurred with a number "out of range" |
| ReferenceError | An error due to an illegal reference |
| SyntaxError | An error due to a syntax error |
| TypeError | An error due to a type error |
| URIError | An error due to encodeURI() |

1. **What are the various statements in error handling**

Below are the list of statements used in an error handling,

* 1. **try:** This statement is used to test a block of code for errors
  2. **catch:** This statement is used to handle the error
  3. **throw:** This statement is used to create custom errors.
  4. **finally:** This statement is used to execute code after try and catch regardless of the result.

1. **What is an event loop**

The Event Loop is a queue of callback functions. When an async function executes, the callback function is pushed into the queue. The JavaScript engine doesn't start processing the event loop until the async function has finished executing the code. **Note:** It allows Node.js to perform non-blocking I/O operations even though JavaScript is single-threaded.

1. **What is call stack**

Call Stack is a data structure for javascript interpreters to keep track of function calls in the program. It has two major actions,

* 1. Whenever you call a function for its execution, you are pushing it to the stack.
  2. Whenever the execution is completed, the function is popped out of the stack.

Let's take an example and it's state representation in a diagram format

function hungry() {

eatFruits();

}

function eatFruits() {

return "I'm eating fruits";

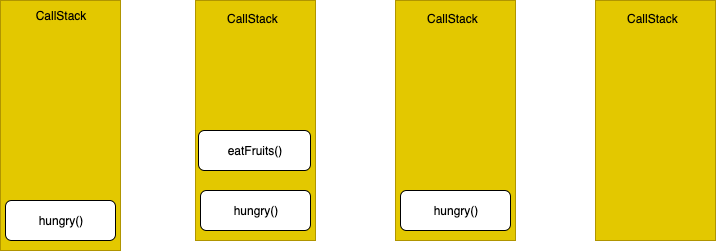
}

// Invoke the `hungry` function

hungry();

The above code processed in a call stack as below,

* 1. Add the hungry() function to the call stack list and execute the code.
  2. Add the eatFruits() function to the call stack list and execute the code.
  3. Delete the eatFruits() function from our call stack list.
  4. Delete the hungry() function from the call stack list since there are no items anymore.

[](https://github.com/sudheerj/javascript-interview-questions/blob/master/images/call-stack.png)

1. **How do you sort elements in an array**

The sort() method is used to sort the elements of an array in place and returns the sorted array. The example usage would be as below,

var months = ["Aug", "Sep", "Jan", "June"];

months.sort();

console.log(months); // ["Aug", "Jan", "June", "Sep"]

1. **How do you reversing an array**

You can use the reverse() method to reverse the elements in an array. This method is useful to sort an array in descending order. Let's see the usage of reverse() method in an example,

let numbers = [1, 2, 5, 3, 4];

numbers.sort((a, b) => b - a);

numbers.reverse();

console.log(numbers); // [1, 2, 3, 4 ,5]

1. **How do you find min and max value in an array**

You can use Math.min and Math.max methods on array variables to find the minimum and maximum elements within an array. Let's create two functions to find the min and max value with in an array,

var marks = [50, 20, 70, 60, 45, 30];

function findMin(arr) {

return Math.min.apply(null, arr);

}

function findMax(arr) {

return Math.max.apply(null, arr);

}

console.log(findMin(marks));

console.log(findMax(marks));

1. **What is a constructor method**

The constructor method is a special method for creating and initializing an object created within a class. If you do not specify a constructor method, a default constructor is used. The example usage of constructor would be as below,

class Employee {

constructor() {

this.name = "John";

}

}

var employeeObject = new Employee();

console.log(employeeObject.name); // John

1. **How do you call the constructor of a parent class**

You can use the super keyword to call the constructor of a parent class. Remember that super() must be called before using 'this' reference. Otherwise it will cause a reference error. Let's the usage of it,

class Square extends Rectangle {

constructor(length) {

super(length, length);

this.name = 'Square';

}

get area() {

return this.width \* this.height;

}

set area(value) {

this.area = value;

}

}

1. **What are the different ways to make an object non-extensible**

You can mark an object non-extensible in 3 ways,

* 1. Object.preventExtensions
  2. Object.seal
  3. Object.freeze

1. **How do you extend classes**

The extends keyword is used in class declarations/expressions to create a class which is a child of another class. It can be used to subclass custom classes as well as built-in objects. The syntax would be as below,

class ChildClass extends ParentClass { ... }

Let's take an example of Square subclass from Polygon parent class,

class Square extends Rectangle {

constructor(length) {

super(length, length);

this.name = 'Square';

}

get area() {

return this.width \* this.height;

}

set area(value) {

this.area = value;

}

}

1. **What are the different methods to find HTML elements in DOM**

If you want to access any element in an HTML page, you need to start with accessing the document object. Later you can use any of the below methods to find the HTML element,

* 1. document.getElementById(id): It finds an element by Id
  2. document.getElementsByTagName(name): It finds an element by tag name
  3. document.getElementsByClassName(name): It finds an element by class name

1. **How to set the cursor to wait**

The cursor can be set to wait in JavaScript by using the property "cursor". Let's perform this behavior on page load using the below function.

function myFunction() {

window.document.body.style.cursor = "wait";

}

and this function invoked on page load

<body onload="myFunction()">

1. **How do you create an infinite loop**

You can create infinite loops using for and while loops without using any expressions. The for loop construct or syntax is better approach in terms of ESLint and code optimizer tools,

for (;;) {}

while(true) {

}

1. **What is the output of below for loops**

for (var i = 0; i < 4; i++) { // global scope

setTimeout(() => console.log(i));

}

for (let i = 0; i < 4; i++) { // block scope

setTimeout(() => console.log(i));

}

The output of the above for loops is 4 4 4 4 and 0 1 2 3

**Explanation:** Due to the event queue/loop of javascript, the setTimeout callback function is called after the loop has been executed. Since the variable i is declared with the var keyword it became a global variable and the value was equal to 4 using iteration when the time setTimeout function is invoked. Hence, the output of the first loop is 4 4 4 4.

Whereas in the second loop, the variable i is declared as the let keyword it becomes a block scoped variable and it holds a new value(0, 1 ,2 3) for each iteration. Hence, the output of the first loop is 0 1 2 3.

1. **What are default parameters**

In E5, we need to depend on logical OR operators to handle default values of function parameters. Whereas in ES6, Default function parameters feature allows parameters to be initialized with default values if no value or undefined is passed. Let's compare the behavior with an examples,

//ES6

var calculateArea = function(height = 50, width = 60) {

return width \* height;

}

console.log(calculateArea()); //300

1. **What are template literals**

Template literals or template strings are string literals allowing embedded expressions. These are enclosed by the back-tick (`) character instead of double or single quotes. In E6, this feature enables using dynamic expressions as below,

var greeting = `Welcome to JS World, Mr. ${firstName} ${lastName}.`

In ES5, you need break string like below,

var greeting = 'Welcome to JS World, Mr. ' + firstName + ' ' + lastName.`

1. **What is destructuring assignment**

The destructuring assignment is a JavaScript expression that makes it possible to unpack values from arrays or properties from objects into distinct variables. Let's get the month values from an array using destructuring assignment

var [one, two, three] = ['JAN', 'FEB', 'MARCH'];

console.log(one); // "JAN"

console.log(two); // "FEB"

console.log(three); // "MARCH"

and you can get user properties of an object using destructuring assignment,

var {name, age} = {name: 'John', age: 32};

console.log(name); // John

console.log(age); // 32

1. **How do you swap variables in destructuring assignment**

If you don't use destructuring assignment, swapping two values requires a temporary variable. Whereas using a destructuring feature, two variable values can be swapped in one destructuring expression. Let's swap two number variables in array destructuring assignment,

var x = 10, y = 20;

[x, y] = [y, x];

console.log(x); // 20

console.log(y); // 10

1. **What are enhanced object literals**

Object literals make it easy to quickly create objects with properties inside the curly braces. For example, it provides shorter syntax for common object property definition as below.

//ES6

var x = 10, y = 20

obj = { x, y }

console.log(obj); // {x: 10, y:20}

1. **What is for...of statement**

The for...of statement creates a loop iterating over iterable objects or elements such as built-in String, Array, Array-like objects (like arguments or NodeList), TypedArray, Map, Set, and user-defined iterables. The basic usage of for...of statement on arrays would be as below,

let arrayIterable = [10, 20, 30, 40, 50];

for (let value of arrayIterable) {

value ++;

console.log(value); // 11 21 31 41 51

}

1. **What is the output of below spread operator array**

[...'John Resig']

The output of the array is ['J', 'o', 'h', 'n', '', 'R', 'e', 's', 'i', 'g'] **Explanation:** The string is an iterable type and the spread operator within an array maps every character of an iterable to one element. Hence, each character of a string becomes an element within an Array.

1. **What is the difference between a parameter and an argument**

Parameter is the variable name of a function definition whereas an argument represents the value given to a function when it is invoked. Let's explain this with a simple function

function myFunction(parameter1, parameter2, parameter3) {

console.log(arguments[0]) // "argument1"

console.log(arguments[1]) // "argument2"

console.log(arguments[2]) // "argument3"

}

myFunction("argument1", "argument2", "argument3")

1. **What is the purpose of some method in arrays**

The some() method is used to test whether at least one element in the array passes the test implemented by the provided function. The method returns a boolean value. Let's take an example to test for any odd elements,

var array = [1, 2, 3, 4, 5, 6 ,7, 8, 9, 10];

var odd = element => element % 2 !== 0;

console.log(array.some(odd)); // true (the odd element exists)

1. **How do you combine two or more arrays**

The concat() method is used to join two or more arrays by returning a new array containing all the elements. The syntax would be as below,

array1.concat(array2, array3, ..., arrayX)

Let's take an example of array's concatenation with veggies and fruits arrays,

var veggies = ["Tomato", "Carrot", "Cabbage"];

var fruits = ["Apple", "Orange", "Pears"];

var veggiesAndFruits = veggies.concat(fruits);

console.log(veggiesAndFruits); // Tomato, Carrot, Cabbage, Apple, Orange, Pears

1. **What is the difference between Shallow and Deep copy**

There are two ways to copy an object,

**Shallow Copy:** Shallow copy is a bitwise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

**Example**

var empDetails = {

name: "John", age: 25, expertise: "Software Developer"

}

to create a duplicate

var empDetailsShallowCopy = empDetails //Shallow copying!

if we change some property value in the duplicate one like this:

empDetailsShallowCopy.name = "Johnson"

The above statement will also change the name of empDetails, since we have a shallow copy. That means we're losing the original data as well.

**Deep copy:** A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.

**Example**

var empDetails = {

name: "John", age: 25, expertise: "Software Developer"

}

Create a deep copy by using the properties from the original object into new variable

var empDetailsDeepCopy = {

name: empDetails.name,

age: empDetails.age,

expertise: empDetails.expertise

}

Now if you change empDetailsDeepCopy.name, it will only affect empDetailsDeepCopy & not empDetails

1. **What is the output of prepend additive operator on falsy values**

If you prepend the additive(+) operator on falsy values(null, undefined, NaN, false, ""), the falsy value converts to a number value zero. Let's display them on browser console as below,

console.log(+null); // 0

console.log(+undefined);// NaN

console.log(+false); // 0

console.log(+NaN); // NaN

console.log(+""); // 0

1. **How do you remove falsy values from an array**

You can apply the filter method on the array by passing Boolean as a parameter. This way it removes all falsy values(0, undefined, null, false and "") from the array.

const myArray = [false, null, 1,5, undefined]

myArray.filter(Boolean); // [1, 5] // is same as myArray.filter(x => x);

1. **How do you get unique values of an array**

You can get unique values of an array with the combination of Set and rest expression/spread(...) syntax.

console.log([...new Set([1, 2, 4, 4, 3])]); // [1, 2, 4, 3]

1. **What is destructuring aliases**

Sometimes you would like to have a destructured variable with a different name than the property name. In that case, you'll use a : newName to specify a name for the variable. This process is called destructuring aliases.

const obj = { x: 1 };

// Grabs obj.x as as { otherName }

const { x: otherName } = obj;

1. **How do you map the array values without using map method**

You can map the array values without using the map method by just using the from method of Array. Let's map city names from Countries array,

const countries = [

{ name: 'India', capital: 'Delhi' },

{ name: 'US', capital: 'Washington' },

{ name: 'Russia', capital: 'Moscow' },

{ name: 'Singapore', capital: 'Singapore' },

{ name: 'China', capital: 'Beijing' },

{ name: 'France', capital: 'Paris' },

];

const cityNames = Array.from(countries, ({ capital}) => capital);

console.log(cityNames); // ['Delhi, 'Washington', 'Moscow', 'Singapore', 'Beijing', 'Paris']

1. **How do you empty an array**

You can empty an array quickly by setting the array length to zero.

let cities = ['Singapore', 'Delhi', 'London'];

cities.length = 0; // cities becomes []

1. **How do you create an array with some data**

You can create an array with some data or an array with the same values using fill method.

var newArray = new Array(5).fill("0");

console.log(newArray); // ["0", "0", "0", "0", "0"]

1. **How do you disable right click in the web page**

The right click on the page can be disabled by returning false from the oncontextmenu attribute on the body element.

<body oncontextmenu="return false;">

1. **How to cancel a fetch request**

Until a few days back, One shortcoming of native promises is no direct way to cancel a fetch request. But the new AbortController from js specification allows you to use a signal to abort one or multiple fetch calls. The basic flow of cancelling a fetch request would be as below,

* 1. Create an AbortController instance
  2. Get the signal property of an instance and pass the signal as a fetch option for signal
  3. Call the AbortController's abort property to cancel all fetches that use that signal For example, let's pass the same signal to multiple fetch calls will cancel all requests with that signal,

const controller = new AbortController();

const { signal } = controller;

fetch("http://localhost:8000", { signal }).then(response => {

console.log(`Request 1 is complete!`);

}).catch(e => {

if(e.name === "AbortError") {

// We know it's been canceled!

}

});

fetch("http://localhost:8000", { signal }).then(response => {

console.log(`Request 2 is complete!`);

}).catch(e => {

if(e.name === "AbortError") {

// We know it's been canceled!

}

});

// Wait 2 seconds to abort both requests

setTimeout(() => controller.abort(), 2000);

1. **What is minimum timeout throttling**

Both browser and NodeJS javascript environments throttles with a minimum delay that is greater than 0ms. That means even though setting a delay of 0ms will not happen instantaneously. **Browsers:** They have a minimum delay of 4ms. This throttle occurs when successive calls are triggered due to callback nesting(certain depth) or after a certain number of successive intervals. Note: The older browsers have a minimum delay of 10ms. **Nodejs:** They have a minimum delay of 1ms. This throttle happens when the delay is larger than 2147483647 or less than 1. The best example to explain this timeout throttling behavior is the order of below code snippet.

function runMeFirst() {

console.log('My script is initialized');

}

setTimeout(runMeFirst, 0);

console.log('Script loaded');

and the output would be in

Script loaded

My script is initialized

If you don't use setTimeout, the order of logs will be sequential.

function runMeFirst() {

console.log('My script is initialized');

}

runMeFirst();

console.log('Script loaded');

and the output is,

My script is initialized

Script loaded

1. **What are the differences between promises and observables**

Some of the major difference in a tabular form

| **Promises** | **Observables** |
| --- | --- |
| Emits only a single value at a time | Emits multiple values over a period of time(stream of values ranging from 0 to multiple) |
| Eager in nature; they are going to be called immediately | Lazy in nature; they require subscription to be invoked |
| Promise is always asynchronous even though it resolved immediately | Observable can be either synchronous or asynchronous |
| Doesn't provide any operators | Provides operators such as map, forEach, filter, reduce, retry, and retryWhen etc |
| Cannot be canceled | Canceled by using unsubscribe() method |

1. **What is babel**

Babel is a JavaScript transpiler to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments. Some of the main features are listed below,

* 1. Transform syntax
  2. Polyfill features that are missing in your target environment (using @babel/polyfill)
  3. Source code transformations (or codemods)

1. **What is the difference between Function constructor and function declaration**

The functions which are created with Function constructor do not create closures to their creation contexts but they are always created in the global scope. i.e, the function can access its own local variables and global scope variables only. Whereas function declarations can access outer function variables(closures) too.

Let's see this difference with an example,

**Function Constructor:**

var a = 100;

function createFunction() {

var a = 200;

return new Function('return a;');

}

console.log(createFunction()()); // 100

**Function declaration:**

var a = 100;

function createFunction() {

var a = 200;

return function func() {

return a;

}

}

console.log(createFunction()()); // 200

1. **What is an observable**

An Observable is basically a function that can return a stream of values either synchronously or asynchronously to an observer over time. The consumer can get the value by calling subscribe() method. Let's look at a simple example of an Observable

import { Observable } from 'rxjs';

const observable = new Observable(observer => {

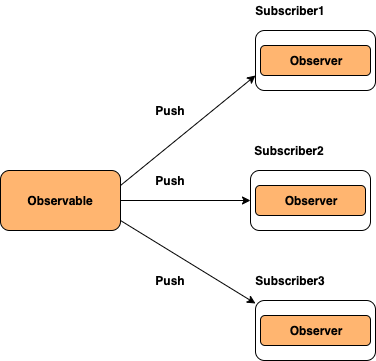
setTimeout(() => {

observer.next('Message from a Observable!');

}, 3000);

});

observable.subscribe(value => console.log(value));

[](https://github.com/sudheerj/javascript-interview-questions/blob/master/images/observables.png)

**Note:** Observables are not part of the JavaScript language yet but they are being proposed to be added to the language

1. **What is the difference between function and class declarations**

The main difference between function declarations and class declarations is hoisting. The function declarations are hoisted but not class declarations.

**Classes:**

const user = new User(); // ReferenceError

class User {}

**Constructor Function:**

const user = new User(); // No error

function User() {

}

1. **What is an async function**

An async function is a function declared with the async keyword which enables asynchronous, promise-based behavior to be written in a cleaner style by avoiding promise chains. These functions can contain zero or more await expressions.

Let's take a below async function example,

async function logger() {

let data = await fetch('http://someapi.com/users'); // pause until fetch returns

console.log(data)

}

logger();

It is basically syntax sugar over ES2015 promises and generators.

1. **What are the differences between arguments object and rest parameter**

There are three main differences between arguments object and rest parameters

* 1. The arguments object is an array-like but not an array. Whereas the rest parameters are array instances.
  2. The arguments object does not support methods such as sort, map, forEach, or pop. Whereas these methods can be used in rest parameters.
  3. The rest parameters are only the ones that haven’t been given a separate name, while the arguments object contains all arguments passed to the function

1. **What are the differences between for...of and for...in statements**

Both for...in and for...of statements iterate over js data structures. The only difference is over what they iterate:

* 1. for..in iterates over all enumerable property keys of an object
  2. for..of iterates over the values of an iterable object.

1. **How do you define instance and non-instance properties**

The Instance properties must be defined inside of class methods. For example, name and age properties defined insider constructor as below,

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

}

But Static(class) and prototype data properties must be defined outside of the ClassBody declaration. Let's assign the age value for Person class as below,

Person.staticAge = 30;

Person.prototype.prototypeAge = 40;

1. **What is the difference between isNaN and Number.isNaN?**
   1. **isNaN**: The global function isNaN converts the argument to a Number and returns true if the resulting value is NaN.
   2. **Number.isNaN**: This method does not convert the argument. But it returns true when the type is a Number and value is NaN.

Let's see the difference with an example,

isNaN(‘hello’); // true

Number.isNaN('hello'); // false

1. **How to invoke an IIFE without any extra brackets?**

Immediately Invoked Function Expressions(IIFE) requires a pair of parenthesis to wrap the function which contains set of statements.

(function(dt) {

console.log(dt.toLocaleTimeString());

})(new Date());

Since both IIFE and void operator discard the result of an expression, you can avoid the extra brackets using void operator for IIFE as below,

void function(dt) {

console.log(dt.toLocaleTimeString());

}(new Date());

1. **Is that possible to use expressions in switch cases?**

You might have seen expressions used in switch condition but it is also possible to use for switch cases by assigning true value for the switch condition. Let's see the weather condition based on temparature as an example,

const weather = function getWeather(temp) {

switch(true) {

case temp < 0: return 'freezing';

case temp < 10: return 'cold';

case temp < 24: return 'cool';

default: return 'unknown';

}

}(10);

1. **What is the easiest way to ignore promise errors?**

The easiest and safest way to ignore promise errors is void that error. This approach is ESLint friendly too.

await promise.catch(e => void e);

1. **What are the different ways to create sparse arrays?**

There are 4 different ways to create sparse arrays in JavaScript

* 1. **Array literal:** Omit a value when using the array literal
  2. const justiceLeague = ['Superman', 'Aquaman', ,'Batman'];

console.log(justiceLeague); // ['Superman', 'Aquaman', empty ,'Batman']

* 1. **Array() constructor:** Invoking Array(length) or new Array(length)
  2. const array = Array(3);

console.log(array); // [empty, empty ,empty]

* 1. **Delete operator:** Using delete array[index] operator on the array
  2. const justiceLeague = ['Superman', 'Aquaman', 'Batman'];
  3. delete justiceLeague[1];

console.log(justiceLeague); // ['Superman', empty, ,'Batman']

* 1. **Increase length property:** Increasing length property of an array
  2. const justiceLeague = ['Superman', 'Aquaman', 'Batman'];
  3. justiceLeague.length = 5;

console.log(justiceLeague); // ['Superman', 'Aquaman', 'Batman', empty, empty]

1. **What is the difference between setTimeout and setImmediate and process.nextTick?**
   1. **Set Timeout:** setTimeout() is to schedule execution of a one-time callback after delay milliseconds.
   2. **Set Immediate:** The setImmediate function is used to execute a function right after the current event loop finishes.
   3. **Process NextTick:** If process.nextTick() is called in a given phase, all the callbacks passed to process.nextTick() will be resolved before the event loop continues. This will block the event loop and create I/O Starvation if process.nextTick() is called recursively.

**Array Methods**

concat, every, forEach, findIndex, map, filter, reduce, some, sort, slice, splice, push, pop, shift, unshift

**JS Problems**

1. **without loop print 1 to 100;**

function myfun(p) {

console.log(p);

return p<100? myfun(p+1):'';

}

1. **Make flat array ( input:** **[1,[2,[3],[4,[5],6],7]] ) (** **output: [1,2,3,4,5,6,7]:)**

let ar = [1,[2,[3],[4,[5],6],7]]

function arrySeq(input, ar=[]) {

input.forEach(e=>{

if(Array.isArray(e)){

arrySeq(e,ar)

} else {

ar.push(e)

}

})

return ar;

}

consle.log(arrySeq(ar))

1. **Write a function for some like (sum(9)(6)())**

function sum(n){

return function(m){

if(isNaN(m)){

return n;

}

return sum(n+m)

}

}

sum(9)(6)() //15

**(or)**

function newadd(...par) {

let val = par?.reduce((a,b)=>a+b);

return function myval(...nval) {

return nval.length==0?val:newadd(...nval, val)

}

}

console.log(add(3)(4)())

console.log(newadd(3,4)(5,6)())

**4) find the number off accuracy iinn string**

--------------------------------------------

let str = "hello data test"

let oc = {}

for(let i=0; i<str.length; i++) {

if(!oc[str[i]]) {

oc[str[i]] = 1;

} else {

oc[str[i]] = 1+oc[str[i]];

}

}

Output: {h:1,e:2,l:2,o:1,d:1,a:2,t:3}

**5) input : let obj = {key:'key1',value:100},{key:'key2',value:200},{key:'key3',value:300}]**

**output: {key1:100, key2:200, key3:3000}**

var obj = [{key:'key1',value:100},{key:'key2',value:200},{key:'key3',value:300}];

let res = obj.reduce((ac={},val)=>{

ac={...ac,[val.key]:val.value};

return ac;

},{})

**6) Remove duplicate element from list**

**input: [1,2,3,4,5,2,3,6]**

**output:[1,4,5,6]**

ar.filter((el,ind)=>ar.indexOf(el)==ar.lastIndexOf(el))

**7)** **input: let object = {name:'hello', id:100, address:{city:'banglore',state:{name:'KA'}},data:{sal:2000,empName:'hello'}}**

**output:{name:'hello', id:100, address.city:'banglore', address.state.name:'KA', data.sal:2000,data.empName:'hello'}**

function requresion(data,res,rootkey) {

res = !res?{}:res;

rootkey=!rootkey?'':rootkey+'.';

for(let k in data) {

var nkey = rootkey+k;

// console.log(k);

typeof data[k] !='object'? res[nkey]=data[k]:requresion(data[k],res,nkey);

}

return res;

}

8)

**output:**

**1-a**

**2-abb**

**3-abbccc**

**4-abbcccdddd**

**5-abbcccddddeeee**

function sq(n){

let ar =['a','b','c','d','e'];

let a = [];

for(let i=0; i<n; i++) {

for(let j=i; j<n; j++) {

a.push(ar[j])

}

}

return a.sort();

}

**9).word reverse with out using split method**

let s = "hello world test";

let resver = "";

var idx;

while(true) {

idx = s.lastIndexOf(" ");

resver+=s.substring(idx)+" ";

if(idx<0)break;

s=s.substring(0,idx)

}

console.log(resver) // ‘test world hello’

**9). Reverse the string?**

let str = ‘Hello’

let out=’’

for(let i=str.length-1; i>=0, i--){

out+=str[i]

}

console.log(out)//’olleH’

**10). Reverse the word**

let word = ‘Hello world’;

let sp = word.split(‘ ’)

let out = ‘’;

for(let i=sp.length-1; i>=0; i--){

out+=sp[i] } console.log(out)//’world Hello’

**11).**

function validate(input) {

let temp = 0;

for(let i=0; i<input.length; i++) {

if(input[i]=='(') {

temp++;

} else if(input=')' && --temp<0) {

// temp--;

return false;

}

}

return temp === 0;

}

const goodBrackets = '())(';

const badBrackets = '(()())))';

const veryBadBrackets = ')(';

**12).** **A simple memorized function to add something**

const memoizedAdd = () => {

let cache = {};

return (n) => {

if (n in cache) {

console.log('Fetching from cache');

return cache[n];

}

else {

console.log('Calculating result');

let result = n + 10;

cache[n] = result;

return result;

}

}

}

const newAdd = memoizedAdd();

console.log(newAdd(9)); // calculated

console.log(newAdd(9)); // cached

**13). Add count for array and display only odd elements**

let ar = ['a','b','c','a','b','a'];

let ob = Object.entries(ar.reduce((a={},i)=>{

a[i]=!a[i]?1:a[i]+1;

return a;

},{})).reduce((e=[],i)=>{

if(i[1]%2!==0)

e=[...e,i[0]];

return e;

},[])

**14). count sequence for each element**

let str = "xxyyzzzklxyzxy";

var current = null;

var cnt = 0;

for (var i = 0; i < str.length; i++) {

if (str[i] != current) {

if (cnt > 0) {

console.log(current + ' comes --> ' + cnt );

}

current = str[i];

cnt = 1;

} else {

cnt++;

}

}

**15). Array sorting without using pre methods single loop**

let arr = [1,2,4,5,3];

for (let j = 0; j < ar.length - 1; j++) {

if (arr[j] > arr[j + 1]) {

let temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

j = -1;

}

}

**16). Given number is Armstrong or not ?**

let input = 153

let k = input;

let r = null;

while(k) {

let p = k%10

r+=p\*p\*p;

k=Math.floor(k/10)

}//1634

//153

console.log(r==k);

//or

function amstrong(input) {

let st = String(input).split('')

let out=0;

st.forEach(e=>{

let p =1;

let n = Number(e);

for(let i=1; i<=st.length; i++) {

p = p\*n

}

out += p

})

return input == out

}

console.log(amstrong(11634))

**17). sequence array create multiple array**

let input = [21, 1, 3, 22, 2, 4, 23, 5,24,6,71,31]

let sr = input.sort((a,b)=>a-b);

let nar = [];

let k = []

for(let i=0; i<sr.length; i++) {

let t = sr[i+1]-sr[i];

if(t===1){

k.push(sr[i]);

}else {

if(k.length>0)

nar.push(k);

k=[]

}

}

console.log(nar)

**18). custom call method**

let obj = {name:'vinay', city:'Banglore'};

Function.prototype.myCall = function(refernce, ...oargs){

let self = this;

self.bind(refernce, ...oargs)()

}

function t(a,b) {

console.log(this.name, '-----',this.city,a,b)

}

t.myCall(obj,1,2);

**19).**

let data = {

"name":"name",

"address":{

"city":{

"value":"karnataka"

}

}

}

function flatObj(obj, res ,key){

res = !res?{}:res;

key=!key?'':key+'.'

for(let k in obj) {

let rootKey = key+k;

if(typeof obj[k]=='object') {

flatObj(obj[k], res ,rootKey)

} else {

res[rootKey] = obj[k]

}

}

return res;

}

function get(data, path, defaultValue) {

let k = flatObj(data)

console.log(k[path]);

}

var res = get(data,'address.city.value','defaultValue');

console.log(res);

**20).**

function per(r) {

let t='';

for(let i=r; i>0; i--){

let s = r-i;

for(let p=1; p<=s; p++) {

t+=' ';

}

for(let k=i; k>=1; k--) {

t+=k;

}

t+='\n'

}

console.log(t)

}

output:

54321

4321

321

21

1

**21).**

function recompose(obj, string) {

var parts = string.split('.');

var newObj = obj[parts[0]];

if (parts[1]) {

parts.splice(0, 1);

var newString = parts.join('.');

return recompose(newObj, newString);

}

return newObj;

}

var obj = { a: { b: '1', c: '2', d:{a:{b:'blah'}}}};

console.log(recompose(obj, 'a.d')); //blah

var getValue = (object, keys) => keys.split('.').reduce((o, k) => (o || {})[k], object);

console.log(getValue({ a: { b: '1', c: '2' } }, 'a'));

console.log(getValue({ a: { b: '1', c: '2' } }, 'foo.bar'));

**22).** **second largetst form list**

function getSecondLargest(nums) {

nums.sort(function(x,y){

return y-x;

});

for(var j=1; j < nums.length; j++)

{

if(nums[j-1] !== nums[j])

{

return nums[j];

}

}

}

getSecondLargest([1,2,3,4,5,5,6,7,6,7,7]);

**23) Remove duplicates form arrays.**var ar = [1,2,3,4,3,2,1];  
var unique\_arr = [];  
for(var i=0; i<ar.length; i++) {  
 if(unique\_arr.indexOf(ar[i]) == -1) {  
 unique\_arr.push(ar[i])  
 }  
}  
or/ var uniqe = new set(ar); it’s will delete the duplicates

or/

var ar = [1,2,3,4,3,2,1];

var un = ar.filter((val,i)=>{ return ar.indexOf(val) == i; })

1. **.Palindrome program?**

var pl = 'cac' ;

var plnd = pl.split("").reverse().join("") ;

if(pl == plnd) { alert(‘true’)}else {alert(‘false’)}

**25.How to get object keys?**

Var object1 = {

a: 'somestring',

b: 42,

c: false

};

console.log(Object.keys(object1));

// expected output: Array ["a", "b", "c"]

Or

for(var x in onject1) {

console.log(x);

}

//expected output a b c

**26. find missing sequence number ?**

Solution1: var s = [1,2,4,6,7,9,11];

for(var i=0; i<s.length-1; i++) {

if(s[i]+1 != s[i+1]) {

console.log(s[i]+1)

}

}

Solution2: var s = [1,2,6,9,13];

for(var i=0; i<s.length; i++) {

var d = s[i+1]-s[i]-1;

for(var k=1; k<=d; k++){

console.log(s[i]+k)

}}

## **27. Rapid Fire**

**Question:** What is typeof []

**Answer:** Object. Actually Array is derived from Object. If you want to check array use Array.isArray(arr)

**Question:** What is typeof arguments

**Answer:** Object. arguments are array like but not array. it has length, can access by index but can't push pop, etc.

**Question:** What is 2+true

**Answer:** 3. The plus operator between a number and a boolean or two boolean will convert boolean to number. Hence, true converts to 1 and you get result of 2+1

**Question:** What is '6'+9

**Answer:** 69. If one of the operands of the plus (+) operator is string it will convert other number or boolean to string and perform a concatenation. For the same reason, "2"+true will return "2true"

**Question:** What is the value of 4+3+2+"1"

**Answer:** 91 . The addition starts from the left, 4+3 results 7 and 7+2 is 9. So far, the plus operator is performing addition as both the operands are number. After that 9 + "1" where one of the operands is string and plus operator will perform concatenation.

**Question:** What is the value of "1"+2+4

**Answer:** "124". For this one "1" + 2 will produce "12" and "12"+4 will generates "124".

**Question:** What is the value of -'34'+10

**Answer:** -24. minus(-) in front of a string is an unary operator that will convert the string to a number and will make it negative. Hence, -'34' becomes, -34 and then plus (+) will perform simple addition as bot

h the operands are number.

**Question:** What is the value of +'dude'

**Answer:** NaN. The plus (+) operator in front of a string is an unary operator that will try to convert the string to number. Here, JavaScript will fail to convert the "dude" to a number and will produce NaN.

**Question:** If you have var y = 1, x = y = typeof x; What is the value of x?

**Answer:** "undefined"

**Question:** for var a = (2, 3, 5); what is the value of a?

**Answer:** 5. The comma operator evaluates each of its operands (from left to right) and retuwrns the value of the last operand. ref: [MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comma_Operator)

**Question:** for var a = (1, 5 - 1) \* 2 what is the value of a?

**Answer:** 8

**Question:** What is the value of !'bang'

**Answer:** false. ! is NOT. If you put ! in front of truthy values, it will return false. Using !! (double bang) is a tricky way to check anything truthy or falsy by avoiding implicit type conversion of == comparison.

**Question:** What is the value of parseFloat('12.3.4')

**Answer:** 12.3

**Question**var t = 9,7,6;

**Answer**Uncaught SyntaxError: Unexpected number

**Question:** What is the value of Math.max([2,3,4,5]);

**Answer:** NaN

**Question:** 3 instanceof Number

**Answer:** false

**Question:**null == undefined

**Answer:** true

**Question:**null === undefined

**Answer:** false

**Question:**[] == []&&[] === []

**Answer:** false

**Question:**What is the value of !!function(){};

**Answer:** true

**Question:** What is the value of typeof bar

**Answer:** "undefined"

**Question:** What is the value of typeof null

**Answer:** "object"

**Question:** If var a = 2, b =3 what would be value of a && b

**Answer:** 3

**Question:** If  10 && 4

Answer : 4

**Question:** What would be consoled var foo = 'outside'; function logIt(){console.log(foo); var foo = 'inside';} logIt();

**Answer:** undefined

**Question:** What is -5%2

**Answer:**-1. the result of remainder always get the symbol of first operand

**Question:** Why .1+.2 != .3

**Answer: true**

**Question: why .1+.2**

**Answer** 0.30000000000000004

**Question:**42..toString()

**Anwser:** "42"

**Question:** 4.2..toString

**Anwser:** //SyntaxError: Unexpected token .

**Question:**42 . toString()

**Anwser:** "42"

**Question:** typeof(NaN)

**Anwser:**"number"

**Question:** 2 in [1,2]

**Anwser:** false. Because "in" returns whether a particular property/index available in the Object. In this case object has index 0 and 1 but don't have 2. Hence you get false.

**Question: 1.1+1.3**

**Anwser:** 2.4000000000000004

**Question: 1.1+1.8**

**Anwser:** 2.9000000000000004

**Question) what is the difference between an Anonymous Function and a named function?**

Anonymous functions exist only after they are called; whereas, Named functions to exist even if not called.

**Question)what is the difference between remove () and removeChild ()?**

The remove() function just removes the element. Whereas, the removeChild() returns the deleted element.

**Question)console.log(1 < 2 < 3);Ans) true**

**console.log(3 > 2 > 1);Ans) fasle**

The first statement returns true which is as expected.The second returns false because of how the engine works regarding operator associativity for < and >. It compares left to right, so 3 > 2 > 1 JavaScript translates to true > 1. true has value 1, so it then compares 1 > 1, which is false

**………..Question)for (let i = 0; i < 5; i++) {**

**setTimeout(function() { console.log(i); }, i \* 1000 );**

**}**

It will print 0 1 2 3 4, because we use **let** instead of **var** here. The variable i is only seen in the for loop’s block scope.

**Question)0.1+0.7**

0.7999999999999999

**0.1+0.2**

0.30000000000000004

**Soulation:** function areTheNumbersAlmostEqual(num1, num2) {

return Math.abs( num1 - num2 ) < Number.EPSILON;

}

console.log(areTheNumbersAlmostEqual(0.1 + 0.2, 0.3));

**Question)**

**console.log(1 + "2" + "2");** Ans) 122

**console.log(1 + +"2" + "2");** Ans) 32

**console.log(1 + -"1" + "2");** Ans) 02

**console.log(+"1" + "1" + "2");** Ans) 112

**console.log( "A" - "B" + "2");** Ans) NaN2

**console.log( "A" - "B" + 2);** Ans) NaN

**28. Each word camel case in given word and it shout reverse order?**

var p = "hello world data";

var ar = p.split(" ");

var cam="";

for(var i=0; i<ar.length; i++) {

var p= ar[i].substr(0,1).toUpperCase()+ar[i].substr(1,ar[i].length);

cam+=p+" ";

cam.trim().split(" ").reverse();

}

**29. given string it should be return only alphabets**

function alphaOnly(a) {

var b = '';

for (var i = 0; i < a.length; i++) {

if (a[i] >= 'A' && a[i] <= 'z') b += a[i];

}

return b;

}

alphaOnly('adDDA904-=-=') // adDDA

**30. Sort array without using sort method ?**

// var data = ["h","g","f","d","s","a"];

var data = [1,3,4,6,7,8,2,0];

var temp =[];

for(var i=0; i<data.length; i++) {

for(var j=i+1; j<data.length; j++) {

if(data[i]>data[j]) {

temp = data[j];

data[j] = data[i];

data[i] = temp;

}

}

}

**31.Febnasie sericesss**

var p1=0;

var p2=1;

var r = 0;

console.log(p1)

console.log(p2)

for(var i=2; i<=5; i++) {

r = p1+p2;

p1=p2;

p2=r;

console.log(r)

}