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 simpliest way to create an empty object is using 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){

var object = {};

object.name=name;

object.age=21;

return object;

}

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 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.

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 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 in an array and 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 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 new array.

1. **What is the purpose of 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 option 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 arrayIntegersOriginal2 = [1, 2, 3, 4, 5];

let arrayIntegersOriginal3 = [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. **How do you compare Object and Map?**

**Objects** are similar to **Maps** in that both let you set keys to values, retrieve those values, delete keys, and detect whether something is stored at a key. Due to this reason, Objects have been used as Maps historically. But there are important differences that make using a Map preferable in certain cases.

1. The keys of an Object are Strings and Symbols, whereas they can be any value for a Map, including functions, objects, and any primitive.
2. The keys in Map are ordered while keys added to object are not. Thus, when iterating over it, a Map object returns keys in order of insertion.
3. You can get the size of a Map easily with the size property, while the number of properties in an Object must be determined manually.
4. A Map is an iterable and can thus be directly iterated, whereas iterating over an Object requires obtaining its keys in some fashion and iterating over them.
5. An Object has a prototype, so there are default keys in the map that could collide with your keys if you're not careful. As of ES5 this can be bypassed by using map = Object.create(null), but this is seldom done.
6. A Map may perform better in scenarios involving frequent addition and removal of key pairs.
7. **What is the difference between == and === operators?**

JavaScript provides both strict(===, !==) and type-converting(==, !=) equality comparison. The strict operators takes type of variable in consideration, while non-strict operators make type correction/conversion based upon values of variables. The strict operators follow the below conditions for different types,

1. Two strings are strictly equal when they have the same sequence of characters, same length, and same characters in corresponding positions.
2. Two numbers are strictly equal when they are numerically equal. i.e, Having the same number value. There are two special cases in this,
   1. NaN is not equal to anything, including NaN.
   2. Positive and negative zeros are equal to one another.
3. Two Boolean operands are strictly equal if both are true or both are false.
4. Two objects are strictly equal if they refer to the same Object.
5. Null and Undefined types are not equal with ===, but equal with ==. i.e, null===undefined --> false but null==undefined --> true

Some of the example 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 function 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 other function as an argument and doesn’t return a function as its return value.

const firstOrder = () => console.log ('Iam a first order functionn!');

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

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

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. Let us take an example of unary function. It stands for single argument accepted by a function.

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

1. **What is 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;

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 re-usability 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 6

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

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

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

As per above code snippets, Push function is impure itself by altering the array and returning an push number index which is independent of parameter value. Whereas Concat on the other hand takes the array and concatenates it with the other array producing a whole new array without side effects. Also, the return value is a concatenation of previous array. Remember that Pure functions are important as they simplify unit testing without any side effects and no need for dependency injection. They also avoid tight coupling and makes harder to break your application by not having any side effects. These principles are coming together with **Immutability** concept of ES6 by giving preference to **const** over **let** usage.

1. **What is the purpose of 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 if block variable 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 | Won't get hoisted |

Let's take an example to see the difference,

function userDetails(username) {

if(username) {

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

console.log(age); // error: age is not defined

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)

}

1. **What is the reason to choose the name let as 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. **How do you redeclare variables in switch block without an error?**

If you try to redeclare variables in a switch block then it will cause errors because there is only one block. For example, the below code block throws a syntax error as below,

let counter = 1;

switch(x) {

case 0:

let name;

break;

case 1:

let name; // SyntaxError for redeclaration.

break;

}

To avoid this error, you can create a nested block inside a case clause will create a new block scoped lexical environment.

let counter = 1;

switch(x) {

case 0: {

let name;

break;

}

case 1: {

let name; // No SyntaxError for redeclaration.

break;

}

}

1. **What is 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. **What is the benefit of using modules?**

There are a lot of benefits to using modules in favour of a sprawling. Some of the benefits are,

* 1. Maintainablity
  2. Reusability
  3. Namespacing

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;

}

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 syntactical 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;

}

}

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

As per the above code, the inner function(greetingInfo) has access to the variables in the outer function scope(Welcome) even after outer function has returned.

1. **What are modules?**

Modules refers small units of independent, reusable code and also act as foundation of many JavaScript design patterns. Most of the JavaScript modules export an object literal, a function, or a constructor

1. **Why do you need modules?**

Below are the list of benefits using modules in javascript ecosystem

* 1. Maintainablity
  2. Reusability
  3. Namespacing

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 background, separate from a web page and provide 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 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. **How do you reuse information across service worker restarts?**

The problem with service worker is that it get terminated when not in use, and restarted when it's next needed, so you cannot rely on global state within a service worker's onfetch and onmessage handlers. In this case, service workers will have access to IndexedDB API in order to persist and reuse across restarts.

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

1. **What is a post message?**

Post message is a method that enables cross-origin communication between Window objects.(i.e, between a page and a pop-up that it spawned, or between a page and an iframe embedded within it). Generally, scripts on different pages are allowed to access each other if and only if the pages follow same-origin policy(i.e, pages share the same protocol, port number, and host).

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, user profile can be stored in a cookie.
  2. Next time the user visits the page, the cookie remembers 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 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 recommended approach than Cookies.

1. **How do you check web storage browser support?**

You need to check browser support for localStorage and sessionStorage before using web storage,

if (typeof(Storage) !== "undefined") {

// Code for localStorage/sessionStorage.

} else {

// Sorry! No Web Storage support..

}

1. **How do you check web workers browser support?**

You need to check browser support for web workers before using it

if (typeof(Worker) !== "undefined") {

// code for Web worker support.

} else {

// Sorry! No Web Worker support..

}

1. **Give an example of web worker?**

You need to follow below steps to start using web workers for counting example

* 1. Create a Web Worker File: You need to write a script to increment the count value. Let's name it as counter.js

let i = 0;

function timedCount() {

i = i + 1;

postMessage(i);

setTimeout("timedCount()",500);

}

timedCount();

Here postMessage() method is used to post a message back to the HTML page 2. Create a Web Worker Object: You can create a web worker object by checking for browser support. Let's name this file as web\_worker\_example.js

if (typeof(w) == "undefined") {

w = new Worker("counter.js");

}

and we can receive messages from web worker

w.onmessage = function(event){

document.getElementById("message").innerHTML = event.data;

};

* 1. Terminate a Web Worker: Web workers will continue to listen for messages (even after the external script is finished) until it is terminated. You can use terminate() method to terminate listening the messages.

w.terminate();

* 1. Reuse the Web Worker: If you set the worker variable to undefined you can reuse the code

w = undefined;

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

WebWorkers don't have access to below javascript objects since they are defined in an external files

* 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 would be as below

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

// promise description

})

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 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 a 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 first function invoking an API call(simulated by setTimeout) and 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 first function and remaining code block get executed. So callbacks used in a way to make sure that certain code doesn’t execute until other code finished 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 server-sent events?**

Server-sent events (SSE) is a server push technology enabling a browser to receive automatic updates from a server via HTTP connection without resorting to polling. These are a one way communications channel - events flow from server to client only. This is been used in Facebook/Twitter updates, stock price updates, news feeds etc.

1. **How do you receive server-sent event notifications?**

The EventSource object is used to receive server-sent event notifications. For example, you can receive messages from server as below,

if(typeof(EventSource) !== "undefined") {

var source = new EventSource("sse\_generator.js");

source.onmessage = function(event) {

document.getElementById("output").innerHTML += event.data + "<br>";

};

}

1. **How do you check browser support for server-sent events?**

You can perform browser support for server-sent events before using it as below,

if(typeof(EventSource) !== "undefined") {

// Server-sent events supported. Let's have some code here!

} else {

// No server-sent events supported

}

1. **What are the events available for server sent events?**

Below are the list of events available for server sent events | Event | Description | |---- | --------- | onopen | It is used when a connection to the server is opened | | onmessage | This event is used when a message is received | | onerror | It happens when an error occurs|

1. **What are the main rules of promise?**

A promise must follow a specific set of rules,

* 1. A promise is an object that supplies a standard-compliant .then() method
  2. A pending promise may transition into either fulfilled or rejected state
  3. A fulfilled or rejected promise is settled and it must not transition into any other state.
  4. Once a promise is settled, the value must not change.

1. **What is callback in callback?**

You can nest one callback inside in another callback to execute the actions sequentially one by one. This is known as callbacks in callbacks.

loadScript('/script1.js', function(script) {

console.log('first script is loaded');

loadScript('/script2.js', function(script) {

console.log('second script is loaded');

loadScript('/script3.js', function(script) {

console.log('third script is loaded');

// after all scripts are loaded

});

})

});

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 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 declare 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 double exclamation?**

The double exclamation or negation(!!) ensures the resulting type is a boolean. If it was falsey (e.g. 0, null, undefined, etc.), it will be false, otherwise, true. For example, you can test IE version using this expression as below,

let isIE8 = false;

isIE8 = !! navigator.userAgent.match(/MSIE 8.0/);

console.log(isIE8); // returns true or false

If you don't use this expression then it returns the original value.

console.log(navigator.userAgent.match(/MSIE 8.0/)); // returns either an Array or null

**Note:** The expression !! is not an operator, but it is just twice of ! operator.

1. **What is the purpose of 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 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, getElementByTagName, createElement etc |

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

The window.history object contains the browsers 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 javascript data types?**

Below are the list of javascript data types available

* 1. Number
  2. String
  3. Boolean
  4. Object
  5. Undefined

1. **What is isNaN?**

The isNaN() function is used to determine whether a value is an illegal number (Not-a-Number) or not. i.e, This function returns true if the value equates to NaN. Otherwise it returns false.

isNaN('Hello') //true

isNaN('100') //false

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 are the problems with global variables?**

The problem with global variables is the conflict of variable names of local and global scope. It is also difficult to debug and test the code that relies on global variables.

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 the purpose of isFinite function?**

The isFinite() function is used to determine whether a number is a finite, legal number. It returns false if the value is +infinity, -infinity, or NaN (Not-a-Number), otherwise it returns true.

isFinite(Infinity); // false

isFinite(NaN); // false

isFinite(-Infinity); // false

isFinite(100); // true

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

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

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

function submit() {

document.form[0].submit();

}

1. **How do you find operating system details?**

The window.navigator object contains information about the visitor's browser os details. Some of the OS properties are avaialble under platform property,

console.log(navigator.platform);

1. **What is the difference between document load and DOMContentLoaded events?**

The DOMContentLoaded event is fired when the initial HTML document has been completely loaded and parsed, without waiting for assets(stylesheets, images, and subframes) to finish loading. Whereas The load event is fired when the whole page has loaded, including all dependent resources(stylesheets, images).

1. **What is the difference between native, host and user objects?**

Native objects are objects that are part of the JavaScript language defined by the ECMAScript specification. For example, String, Math, RegExp, Object, Function etc core objects defined in the ECMAScript spec. Host objects are objects provided by the browser or runtime environment (Node). For example, window, XmlHttpRequest, DOM nodes etc considered as host objects. User objects are objects defined in the javascript code. For example, User object created for profile information.

1. **What are the tools or techniques used for debugging JavaScript code?**

You can use below tools or techniques for debugging javascript

* 1. Chrome Devtools
  2. debugger statement
  3. Good old console.log statement

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 difference between an attribute and a property?**

Attributes are defined on the HTML markup whereas properties are defined on the DOM. For example, the below HTML element has 2 attributes type and value,

<input type="text" value="Name:">

You can retrieve the attribute value as below,

const input = document.querySelector('input');

console.log(input.getAttribute('value')); // Good morning

console.log(input.value); // Good morning

And after you change the value of the text field to "Good evening", it becomes like

console.log(input.getAttribute('value')); // Good morning

console.log(input.value); // Good evening

1. **What is same-origin policy?**

The same-origin policy is a policy that prevents JavaScript from making requests across domain boundaries. An origin is defined as a combination of URI scheme, hostname, and port number. If you enable this policy then it prevents a malicious script on one page from obtaining access to sensitive data on another web page using Document Object Model(DOM).

1. **What is the purpose of void 0?**

Void(0) is used to prevent the page from refreshing. This will be helpful to eliminate the unwanted side-effect, because it will return the undefined primitive value. It is commonly used for HTML document that uses href="JavaScript:Void(0);" within an element. i.e, when you click a link, the browser loads a new page or refreshes the same page. But this behavior will be prevented using this expression. For example, the below link notify the message without reloading the page

<a href="JavaScript:void(0);" onclick="alert('Well done!')">Click Me!</a>

1. **Is JavaScript a compiled or interpreted language?**

JavaScript is an interpreted language, not a compiled language. An interpreter in the browser reads over the JavaScript code, interprets each line, and runs it. Nowadays modern browsers use a technology known as Just-In-Time (JIT) compilation, which compiles JavaScript to executable bytecode just as it is about to run.

1. **Is JavaScript a case-sensitive language?**

Yes, JavaScript is a case sensitive language. The language keywords, variables, function & object names, and any other identifiers must always be typed with a consistent capitalization of letters.

1. **Is there any relation between Java and JavaScript?**

No, they are entirely two different programming languages and has nothing to do with each other. But both of them are Object Oriented Programming languages and like many other languages, they follow similar syntax for basic features(if, else, for, switch, break, continue etc).

1. **What are events?**

Events are "things" that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can react on these events. Some of the examples of HTML events are,

* 1. Web page has finished loading
  2. Input field was changed
  3. Button was clicked

Let's describe the behavior of click event for button element,

<!doctype html>

<html>

<head>

<script>

function greeting() {

alert('Hello! Good morning');

}

</script>

</head>

<body>

<button type="button" onclick="greeting()">Click me</button>

</body>

</html>

1. **Who created javascript?**

JavaScript was created by Brendan Eich in 1995 during his time at Netscape Communications. Initially it was developed under the name Mocha, but later the language was officially called LiveScript when it first shipped in beta releases of Netscape.

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 hyper link are some common usecases.

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 are the steps involved in return false usage?**

The return false statement in event handlers performs the below steps,

* 1. First it stops the browser's default action or behaviour.
  2. It prevents the event from propagating the DOM
  3. Stops callback execution and returns immediately when called.

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 window. The Browser Object Model is not standardized and can change based on different browsers.

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

The setTimeout() method is used to call a function or evaluates 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 evaluates 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. **Why is JavaScript treated as Single threaded?**

JavaScript is a single-threaded language. Because the language specification does not allow the programmer to write code so that the interpreter can run parts of it in parallel in multiple threads or processes. Whereas languages like java, go, C++ can make multi-threaded and multi-process programs.

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 are the syntax rules of JSON?**

Below are the list of syntax rules of JSON

* 1. The data is in name/value pairs
  2. The data is separated by commas
  3. Curly braces hold objects
  4. Square brackets hold arrays

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 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 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 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 redirect new page in javascript?**

In vanilla javascript, you can redirect to a new page using location property of window object. The syntax would be as follows,

function redirect() {

window.location.href = 'newPage.html';

}

1. **How do you check whether a string contains a substring?**

There are 3 possible ways to check whether a string contains a substring or not,

* 1. **Using includes:** ES6 provided String.prototype.includes method to test a string contains a substring

var mainString = "hello", subString = "hell";

mainString.includes(subString)

* 1. **Using indexOf:** In an ES5 or older environments, you can use String.prototype.indexOf which returns the index of a substring. If the index value is not equal to -1 then it means the substring exist in the main string.

var mainString = "hello", subString = "hell";

mainString.indexOf(subString) !== -1

* 1. **Using RegEx:** The advanced solution is using Regular expression's test method(RegExp.test), which allows for testing for against regular expressions

var mainString = "hello", regex = "/hell/";

regex.test(mainString)

1. **How do you validate an email in javascript?**

You can validate an email in javascript using regular expressions. It is recommended to do validations on the server side instead client side. Because the javascript can be disabled on the client side.

function validateEmail(email) {

var re = /^(([^<>()**\[\]\\**.,;:\s@"]+(**\.**[^<>()**\[\]\\**.,;:\s@"]+)\*)|(".+"))@((**\[**[0-9]{1,3}**\.**[0-9]{1,3}**\.**[0-9]{1,3}**\.**[0-9]{1,3}**\]**)|(([a-zA-Z\-0-9]+**\.**)+[a-zA-Z]{2,}))$/;

return re.test(String(email).toLowerCase());

}

The above regular expression regular accepts unicode characters.

1. **How do you get the current url with javascript?**

You can use window.location.href expression to get the current url path and you can use the same expression for updating the URL too. You can also use document.URL for read-only purpose but this solution has issues in FF.

console.log('location.href', window.location.href); // Returns full URL

1. **What are the various url properties of location object?**

The below Location object properties can be used to access URL components of the page,

* 1. href - The entire URL
  2. protocol - The protocol of the URL
  3. host - The hostname and port of the URL
  4. hostname - The hostname of the URL
  5. port - The port number in the URL
  6. pathname - The path name of the URL
  7. search - The query portion of the URL
  8. hash - The anchor portion of the URL

1. **How do get query string values in javascript?**

You can use URLSearchParams to get query string values in javascript. Let's see an example to get the client code value from URL query string,

const urlParams = new URLSearchParams(window.location.search);

const clientCode = urlParams.get('clientCode');

1. **How do you check if a key exists in an object?**

You can check whether a key exists in an object or not using two approaches,

* 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 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. **What is an arguments object?**

The arguments object is an Array-like object accessible inside functions that contains the values of the arguments passed to that function. For example, let's see how to use arguments object inside sum function,

function sum() {

var total = 0;

for (var i = 0, len = arguments.length; i < len; ++i) {

total += arguments[i];

}

return total;

}

sum(1, 2, 3) // returns 6

1. **How do you make first letter of the string in an uppercase?**

You can create a function which uses chain of string methods such as charAt, toUpperCase and slice methods to generate a string with first letter in uppercase.

function capitalizeFirstLetter(string) {

return string.charAt(0).toUpperCase() + string.slice(1);

}

1. **What are the pros and cons of for loop?**

The for-loop is a commonly used iteration syntax in javascript. It has both pros and cons **Pros**

* 1. Works on every environment
  2. You can use break and continue flow control statements **Cons**
  3. Too verbose
  4. Imperative
  5. You might face one-by-off errors

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 use date.getTime() method to compare date values instead comparision 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 check if a string starts with another string?**

You can use ECMAScript 6's String.prototype.startsWith() method to check a string starts with another string or not. But it is not yet supported in all browsers. Let's see an example to see this usage,

"Good morning".startsWith("Good"); // true

"Good morning".startsWith("morning"); // 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 begining 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. **Is the !-- notation represents a special operator?**

No,that's not a special operator. But it is a combination of 2 standard operators one after the other,

* 1. A logical not (!)
  2. A prefix decrement (--)

At first, the value decremented by one and then tested to see if it is equal to zero or not for determining the truthy/falsy value.

1. **How do you assign default values to variables?**

You can use the logical or operator || in an assignment expression to provide a default value. The syntax looks like as below,

var a = b || c;

As per the above expression, variable 'a 'will get the value of 'c' only if 'b' is falsy (if is null, false, undefined, 0, empty string, or NaN), otherwise 'a' will get the value of 'b'.

1. **How do you define multiline strings?**

You can define multiline string literals using '' character followed by line terminator.

var str = "This is a \

very lengthy \

sentence!";

But if you have a space after the '' character, the code will look exactly the same, but it will raise a SyntaxError.

1. **What is an app shell model?**

An application shell (or app shell) architecture is one way to build a Progressive Web App that reliably and instantly loads on your users' screens, similar to what you see in native applications. It is useful for getting some initial HTML to the screen fast without a network.

1. **Can we define properties for functions?**

Yes, We can define properties for functions because functions are also objects.

fn = function(x) {

//Function code goes here

}

fn.name = "John";

fn.profile = function(y) {

//Profile code goes here

}

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 are break and continue statements?**

The break statement is used to "jumps out" of a loop. i.e, It breaks the loop and continues executing the code after the loop.

for (i = 0; i < 10; i++) {

if (i === 5) { break; }

text += "Number: " + i + "<br>";

}

The continue statement is used to "jumps over" one iteration in the loop. i.e, It breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

for (i = 0; i < 10; i++) {

if (i === 5) { continue; }

text += "Number: " + i + "<br>";

}

1. **What are js labels?**

The label statement allows us to name loops and blocks in JavaScript. We can then use these labels to refer back to the code later. For example, the below code with labels avoids printing the numbers when they are same,

var i, j;

loop1:

for (i = 0; i < 3; i++) {

loop2:

for (j = 0; j < 3; j++) {

if (i === j) {

continue loop1;

}

console.log('i = ' + i + ', j = ' + j);

}

}

// Output is:

// "i = 1, j = 0"

// "i = 2, j = 0"

// "i = 2, j = 1"

1. **What are the benefits of keeping declarations at the top?**

It is recommended to keep all declarations at the top of each script or function. The benefits of doing this are,

* 1. Gives cleaner code
  2. It provides a single place to look for local variables
  3. Easy to avoid unwanted global variables
  4. It reduces the possibility of unwanted re-declarations

1. **What are the benefits of initializing variables?**

It is recommended to initialize variables because of the below benefits,

* 1. It gives cleaner code
  2. It provides a single place to initialize variables
  3. Avoid undefined values in the code

1. **What are the recommendations to create new object?**

It is recommended to avoid creating new objects using new Object(). Instead you can initialize values based on it's type to create the objects.

* 1. Assign {} instead of new Object()
  2. Assign "" instead of new String()
  3. Assign 0 instead of new Number()
  4. Assign false instead of new Boolean()
  5. Assign [] instead of new Array()
  6. Assign /()/ instead of new RegExp()
  7. Assign function (){} instead of new Function()

You can define them as an example,

var v1 = {};

var v2 = "";

var v3 = 0;

var v4 = false;

var v5 = [];

var v6 = /()/;

var v7 = function(){};

1. **How do you define JSON arrays?**

JSON arrays are written inside square brackets and array contain javascript objects. For example, the JSON array of users would be as below,

"users":[

{"firstName":"John", "lastName":"Abrahm"},

{"firstName":"Anna", "lastName":"Smith"},

{"firstName":"Shane", "lastName":"Warn"}

]

1. **How do you generate random integers?**

You can use Math.random() with Math.floor() to return random integers. For example, if you want generate random integers between 1 to 10, the multiplication factor should be 10,

Math.floor(Math.random() \* 10) + 1; // returns a random integer from 1 to 10

Math.floor(Math.random() \* 100) + 1; // returns a random integer from 1 to 100

**Note:** Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)

1. **Can you write a random integers function to print integers with in a range?**

Yes, you can create a proper random function to return a random number between min and max (both included)

function randomInteger(min, max) {

return Math.floor(Math.random() \* (max - min + 1) ) + min;

}

randomInteger(1, 100); // returns a random integer from 1 to 100

randomInteger(1, 1000); // returns a random integer from 1 to 1000

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 few MBs, but by tree shaking it can bring down the size to just few hundred KBs. Tree shaking is been implemented in Rollup and Webpack bunders.

**What is the use of isNaN function?**

isNan function returns true if the argument is not a number otherwise it is false.

**6. What is negative infinity?**

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

**What are JavaScript Data Types?**

Following are the JavaScript Data types:

* Number
* String
* Boolean
* Object
* Undefined
* **What are undeclared and undefined variables?**
* Undeclared variables are those that do not exist in a program and are not declared. If the program tries to read the value of an undeclared variable, then a runtime error is encountered.
* Undefined variables are those that are declared in the program but have not been given any value. If the program tries to read the value of an undefined variable, an undefined value is returned.

**Write the code for adding new elements dynamically?**

* <html>
* <head>
* <title>t1</title>
* <script type="text/javascript">
* function addNode() { var newP = document.createElement("p");
* var textNode = document.createTextNode(" This is a new text node");
* newP.appendChild(textNode); document.getElementById("firstP").appendChild(newP); }
* </script> </head>
* <body> <p id="firstP">firstP<p> </body>
* </html>

**What are global variables? How are these variable declared and what are the problems associated with using them?**

Global variables are those that are available throughout the length of the code, that is, these have no scope. The var keyword is used to declare a local variable or object. If the var keyword is omitted, a global variable is declared.

Example:

// Declare a global globalVariable = "Test";

The problems that are faced by using global variables are the clash of variable names of local and global scope. Also, it is difficult to debug and test the code that relies on global variables.

**What is 'this' keyword in JavaScript?**

'This' keyword refers to the object from where it was called.

**Explain the working of timers in JavaScript? Also elucidate the drawbacks of using the timer, if any?**

Timers are used to execute a piece of code at a set time or also to repeat the code in a given interval of time. This is done by using the functions **setTimeout, setInterval**and**clearInterval**.

The **setTimeout(function, delay)** function is used to start a timer that calls a particular function after the mentioned delay. The **setInterval(function, delay)** function is used to repeatedly execute the given function in the mentioned delay and only halts when cancelled. The **clearInterval(id)**function instructs the timer to stop.

Timers are operated within a single thread, and thus events might queue up, waiting to be executed.

**What is called Variable typing in Javascript?**

Variable typing is used to assign a number to a variable and the same variable can be assigned to a string.

Example

i = 10;

i = "string";

This is called variable typing.

**What do mean by NULL in Javascript?**

The NULL value is used to represent no value or no object. It implies no object or null string, no valid boolean value, no number and no array object.

**What is the function of delete operator?**

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

Example

var student= {age:20, batch:"ABC"};

delete student.age;

**What is an undefined value in JavaScript?**

Undefined value means the

* Variable used in the code doesn't exist
* Variable is not assigned to any value
* Property doesn't exist

**What is the use of Void(0)?**

* Void(0) is used to prevent the page from refreshing and parameter "zero" is passed while calling.
* Void(0) is used to call another method without refreshing the page.
* **How can a page be forced to load another page in JavaScript?**
* The following code has to be inserted to achieve the desired effect:
* <script language="JavaScript" type="text/javascript" >
* <!-- location.href="http://newhost/newpath/newfile.html"; //--></script>
* **What is unshift method in JavaScript?**
* Unshift method is like push method which works at the beginning of the array. This method is used to prepend one or more elements to the beginning of the array.
* **How will you explain closures in JavaScript? When are they used?**
* Closure is a locally declared variable related to a function which stays in memory when the function has returned.
* For example:
* function greet(message) {
* console.log(message);
* }
* function greeter(name, age) {
* return name + " says howdy!! He is " + age + " years old";
* }
* // Generate the message
* var message = greeter("James", 23);
* // Pass it explicitly to greet
* greet(message);
* This function can be better represented by using closures
* function greeter(name, age) {
* var message = name + " says howdy!! He is " + age + " years old";
* return function greet() {
* console.log(message);
* };
* }
* // Generate the closure
* var JamesGreeter = greeter("James", 23);
* // Use the closure
* JamesGreeter();

**Explain the for-in loop?**

* The for-in loop is used to loop through the properties of an object.
* The syntax for the for-in loop is -
* for (variable name in object){
* statement or block to execute
* }
* In each repetition, one property from the object is associated to the variable name, and the loop is continued till all the properties of the object are depleted.

**Define event bubbling?**

JavaScript allows DOM elements to be nested inside each other. In such a case, if the handler of the child is clicked, the handler of parent will also work as if it were clicked too.

**Write about the errors shown in JavaScript?**

JavaScript gives a message if it encounters an error. The recognized errors are -

* Load-time errors: The errors shown at the time of the page loading are counted under Load-time errors. These errors are encountered by the use of improper syntax, and thus are detected while the page is getting loaded.
* Run-time errors: This is the error that comes up while the program is running. It is caused by illegal operations, for example, division of a number by zero, or trying to access a non-existent area of the memory.
* Logic errors: It is caused by the use of syntactically correct code, which does not fulfill the required task. For example, an infinite loop.