Java Script Interview Questions

Q.1) What is "undefined" in JavaScript ?

As the name suggests, undefined means "not defined".

So we declare a variable but do not assign a value to it, the variable becomes undefined.

Undefine is a datatype as well in javascript.

Example

let x;

console.log(x);

console.log(typeof x);

Explanation:

In up code I declare a variable x but not define so it automatically becomes undefine, and it’s type Is also undefine.

Q.2) What is "null" in JavaScript ?

Null is exactly same as other programming language which means “nothing”.

When you have a variable or object which you want to make empty then you assign “null” to a variable or an object.

let x=null;

console.log(x);

explanation

We explicitly assign null to a variable.

let x=null;

let y;

console.log(x==y); // true

console.log(x===y); // false

explanation

If we compare x and y with == then it gives true because both have nothing.

But when we check with === it gives false because === also check data type and null, undefine are not same datatype

Q.2) What will be the output of undefined==null & undefined===null ? Why ?

output of undefined==null true

output of undefined===null false

explanation

If we compare x and y with == then it gives true because both have nothing.

But when we check with === it gives false because === also check data type and null, undefine are not same datatype

Null is a primitive datatype.

Q.3) Can you explicitly assign "undefined" to a variable ? (let i = undefined)

Yes you can explicitly assign "undefined" to a variable because undefine is also keyword.

let x = undefined;

console.log(x);

Q.4) What is hoisting in javascript ?

Hosting is a concept in javascript in which we can use variable or function before they're declared.

Example

console.log(foo);

var foo = 'foo';

Explanation

in up code foo is define after console.log so it’s output is “undefine” because of hosting.

Javascript internally create the variable foo before initialization.

This is because the JavaScript interpreter splits the declaration and assignment of functions and variables:

It "hoists" your declarations to the top of their containing scope before execution.

Q.5) What is the scope of a variable ?

Scope determines the accessibility (visibility) of variables.

**Types of Scopes in JavaScript:**

* Block scope
* Function scope
* Local scope
* Global scope

Block Scope

Before ES6 (2015), JavaScript had only **Global Scope** and **Function Scope**.

ES6 introduced two important new JavaScript keywords: let and const.

These two keywords provide **Block Scope** in JavaScript.

Variables declared inside a { } block cannot be accessed from outside the block:

**let keyword:**

**Example:**

{

  let x = 2;

 }

 x cannot be used here

**var keyword:**

**Example:**

{

  var x = 2;

 }

 x can be used here

note : Variables declared with the*var* keyword cannot have block scope and they can be declared inside a { } block and can be accessed from outside the block.

**Local scope:**

Variables declared inside a function become local to the function.

Local variables are created when a function starts and deleted when the function is executed.

**Local variables have Function Scope which means that they can only be accessed from within the function.**

Example

function foo() {

  var x = '1';

  console.log('inside function: ', x);

}

foo();          // Inside function: 1

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

explanation:

We create a function foo and inside the function we declare a variable ‘x’. If we access the x outside of the function it gives error because the scope of the variable is local. It access only inside the function.

## Function Scope

JavaScript has function scope: Each function creates a new scope.

Variables defined inside a function are not accessible (visible) from outside the function.

Variables declared with var, let and const are quite similar when declared inside a function.

They all have **Function Scope**:

function myFunction() {

  var carName = "Volvo";   // Function Scope

}

function myFunction() {

  let carName = "Volvo";   // Function Scope

}

function myFunction() {

  const carName = "Volvo";   // Function Scope

}

**Global scope:**

Variables declared Globally (outside of any function) have Global Scope. Global variables can be accessed from anywhere in a program.

Similar to function scope variables declared with **var**,**let**and**const**are quite similar when declared outside a block.

Example

// Global scope

var x = '1'

const y = '2'

let z = '3'

console.log(x);    // 1

console.log(y);    // 2

console.log(z);    // 3

function getNo() {

    console.log(x);   // x is accessible here

    console.log(y);   // y is accessible here

    console.log(z);   // z is accessible here

}

getNo();

explanation

Variable which are declare outside of the function are global variable.

Q.6) How does block scope work ?

This scope restricts the variable that is declared inside a specific block, from access by the outside of the block.

The *let*& *const* keyword facilitates the variables to be block scoped.

In order to access the variables of that specific block, we need to create an object for it.

Variables declared with the *var*keyword, do not have block scope.

Example

{

  let p = 110;

  const q = 111;

}

console.log(p); // Uncaught ReferenceError: p is not defined

console.log(q); // Uncaught ReferenceError: q is not defined

Explanation

we create a block inside it define a variable using let, const. If we access that variables outside of the block it gives error. You can see in example.

Q.7) Should you terminate all lines by a ‘;’ ?

The answer is Yes. It is an optional but it is a good practice to terminate all the lines.

Javascript automatic terminate the line by ‘;’ at the end of the line.

But in some cases it won’t.

Example

let a  = 4 \*

5

console.log(a);

explanation

in upper code js know that after ‘ \* ’ sign there must b a value so it doesn’t terminate the line by a ‘;’

Q.8) Why this code of line returning ‘undefined’ ?

function test(){

  return

{

    a:5

  }

}

const obj = test();

console.log(obj);

explanation

When the upper code is execute the interpreter check line by line and when it reaches to return keyword and not found ‘{‘ braces just by return it terminate by ‘;’. So our output comes undefined.

If we want to get the output ‘5’ we have to write like

return {   // start the braces just by return keyword not from next line

  a:5

}

In this way the interpreter go inside the return and give the output 5.

Q.9) What is Rest operator in javascript ?

The operator is used to put some user-supplied values into an Array.

The text after the rest operator references the values you wish to encase inside an array.

You can only use it before the last parameter in a function definition.

We always use ‘ …’ in last parameter.

OR

The rest parameter (...) allows a function to treat an indefinite number of arguments as an array.

Rest parameter is added in ES2015 or ES6 which improved the ability to handle parameter.

**Note:** When … is at the end of the function parameter, it is the rest parameter. It stores n number of parameters as an array. Let’s see how the rest parameter works:

Example

function sumAll(...args) {

  console.log(args)

 }

 // here the function can be called with any number of arguments

 sumAll(1)

 sunAll(1,2,3)

function khan(a,b,…c){ // if we have more than 1 parameter value than always     use rest operator in last value

  console.log(a);

console.log(b);

console.log(c);

  }

  Khan(1,2,45,67,89);

explanation

basically when we use … in parameter it means the variable takes n numbers of values and create an Array

output.

[1]

[1,2,3]

Q.9) What is Spread operator in javascript ?

while using rest operator we make an array of parameters , but sometimes we need to the exact opposite and extract all the values from array or object , this is where we use spread operator

It is commonly used for merging arrays, cloning objects, and passing arguments to functions.

The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

Example

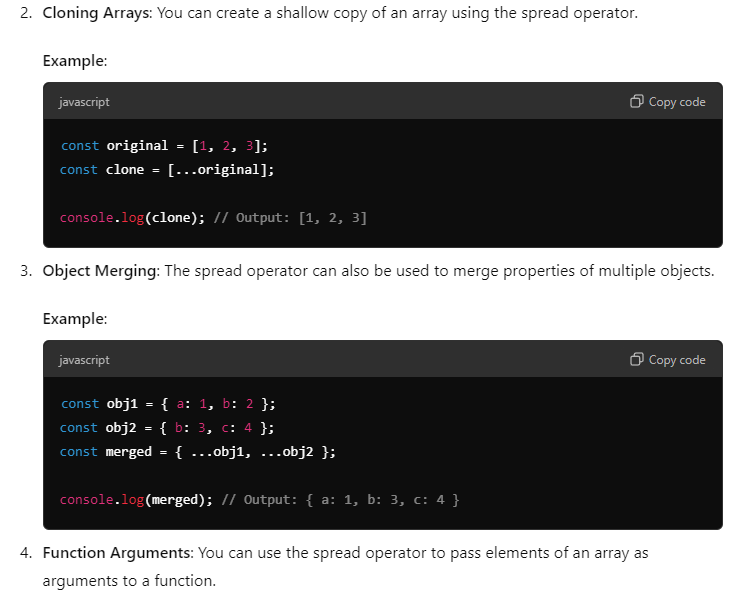
const numbersOne = [1, 2, 3];

const numbersTwo = [4, 5, 6];

const numbersCombined = [...numbersOne, ...numbersTwo];

explanation

In upper code numberOne, numberTwo both the array are assign to numberCombined using spread Operator.



A screenshot of a computer

Description automatically generated

Question related to REST and Spread Operator

Q.10) Can we use argument object in arrow function?

No we can not use argument object with arrow function. Always use normal function

Example 1.

function sum(){

  console.log(arguments);

}

sum(1,2);

sum(2,3,4);

in this way we use argument object

Example 2.

const sum = ()=>{

  console.log(arguments);

}

sum(1,2);

sum(2,3,4);

this is an arrow function. If we use like this it gives error.

Q.b) Which is the best way to create a new array with assignments?

The best way to create a new array is to use spread operator not assignment operator.

Example

arr1 == arr2 is not good method because both are objects.

So use spread operator

let arr1 = [1,2,3,4,5];

let arr2 = [...arr1,6,7,8,9];

console.log(arr2)

output

[1,2,3,4,5,6,7,8,9]

Q.11) How can you handle the “n” number of parameters passed to a function ? or create a function which can take any number of parameters and returns sum or max or any other such calculative values.

Exapmle

const sum = (...val)=>{

console.log(Math.max(...val)); // one by one it pass value in max function

                                //and function give the max value

}

sum(2,3,4);

Explanation

Here we pass n number of values to function parameter so we use rest operator.

Now, inside function we calculate max value so we don’t write without rest parameter otherwise it gives NAN which means not a number

Q.12) Can the rest operator be placed anywhere in the function parameter list? Or

function test(...a,b){

//statements

}

Is this piece of code valid? Give the reason.

No we can not use rest parameter in the beginning of the parameter value. If the parameter value is single then we can use otherwise we always use with last value.

Upper function is not correct because it use rest operator with first value.

Q.12) What is Infinity in JavaScript?

It is a special integer value which is grater than any value or in other words it is the highest possible value available in JavaScript.

There are 2 possibilities of Infinity –

1. Positive Infinity (POSITIVE\_INFINITY)

Example

Console.log(Number. POSITIVE\_INFINITY)

1. Negative Infinity (NEGATIVE\_INFINITY)

Example

Console.log(Number. POSITIVE\_INFINITY)

Q.13 ) Why we get infinity in javascript?

In JavaScript the numbers are store in 64-bit format.

It means the value which cannot fit in 64-bit format will return infinity.

Q.14) Where we use Infinity?

Ideally we expect that in some calculation you might get some huge value and you want put some validation than it is useful.

Example

if(num === Number.POSITIVE\_INFINITY){

  return "Positive Integer"

}else if(num === Number.NEGATIVE\_INFINITY){

  return "Negative Integer";

}else{

  return num   // some calculation

}

Explanation

Suppose you are calculating any value and you know the result is huge. In this case we use Infinity.

If result value comes under 64-bit formate than ‘else’ part run otherwis ‘if or else if part will run’.

Q.15) What is the output of the code?

Var a = 1/0;

Console.log(a);

The answer is Infinity.

In other programming language it gives error but in js it gives infinity.

Q.16) What is NAN and when we get this error in JavaScript?

NaN is a global property which represent Not- A -Number.

We normally get this error when there is a non-numeric value or operation performed.

Example

let a = 5;

let b = "abdul";

console.log(a+b);

Explanation

In upper code I add a and b but b is a string so the output is NAN.

Note: We can not check NaN == NaN because every time NAN has a unique value so use isNAN() method.

The most preferable method which check NaN and also checks Infinity or -Infinity is

isFinite() method.

In sort isFinite method check the number is regular or not.

Example

let a = 5;

let b = "abdul";

console.log(isFinite(a+b)); // false

console.log(isFinite(a+5)); // True

console.log(isFinite(Number.MAX\_VALUE \* 4)); // false

console.log(isFinite(Infinity)); //false

explanation

In second console the number is regular so it returns true and all others are not regular so its return false.

Arrow Function

**Arrow function {()=>}**is concise way of writing Javascript functions in shorter way.

**Arrow functions** were introduced in the ES6 version.

**Arrow functions** are anonymous functions i.e. functions without a name and are not bound by an identifier.

They make our code more structured and readable.

Arrow functions do not return any value and can be declared without the function keyword.

They are also called **Lambda Functions**.

const gfg = () => {

  console.log( "Hi Geek!" );

}

## ****Arrow Function with Parameters****

const gfg = ( x, y, z ) => {

  console.log( x + y + z )

}

gfg( 10, 20, 30 );

## ****Arrow Function with IIFE(Immediately Invoked Arrow Function)****

IIAF

(function (){

  // peice of code

})();

IIAF with arrow function

(()=>{

  // peice of code

})();

**Advantages of Arrow Functions**

* Arrow functions reduce the size of the code.
* The return statement and function brackets are optional for single-line functions.
* It increases the readability of the code.
* Arrow functions provide a lexical this binding. It means, they inherit the value of “this” from the enclosing scope. This feature can be advantageous when dealing with event listeners or callback functions where the value of “this” can be uncertain.

**Limitations of Arrow Functions**

* Arrow functions do not have the prototype property.
* Arrow functions cannot be used with the new keyword.
* Arrow functions cannot be used as constructors.
* These functions are anonymous and it is hard to debug the code.
* Arrow functions cannot be used as generator functions that use the yield

Q.17) Is the Keyword ‘this’ refers in the arrow function?

In this case keyword ‘this’ refers to globally.

Exapmple 1

const test = ()=>{

  console.log(this);

}

test();

example 2

If we create object with normal function and see the behaviour of this keyword

const obj = {

  test(){

      console.log(this)

  }

}

obj.test();

output

{ test: [Function: test] } // it refers the test method of the object

Example 3

If we create object with normal function and see the behaviour of this keyword

const obj = {

  test:()=>{

      console.log(this)

  }

}

obj.test();

output

Window {…..} // refers globally

Reason:

In ES5 the function also behave like a class accord to function call on which way we call the function. Like if we call the function with ‘new’ keyword it behaves like a function.

But in ES6 function don’t have any job to behave like a class so this always refers globally.

Q.18) Explain the syntactical features of arrow function?

1

const sum = a => {

  console.log(a);

}

2

const sum = a => return console.log(a);

sum(10);

3

const sum = (a,b) => {

  return a+b;

}

console.log(sum(5,7));

# Closure in JavaScript

A closure is a feature of JavaScript that allows inner functions to access the outer function variables/parameter as well as the global variables.

Closure is useful when you want to make few private members available globally when needed.

Example

const outerFun = (a) => {

  let b=5; //Global variable for inner function

  const inner = ()=>{

      let sum = a+b;

      console.log(sum)

  }

  return inner;

}

let innerF = outerFun(5)

innerF();

explanation

According to definition I create a Outer function which have a variable, get a parameter and also inner function when we call outer function and return inner function not call to inner function then inner function access the variable and parameter value of outer function because of environment.

Q.19) How can you access private variable or function outside of the scope?

We can access with the help of clouser.

const outerFun = (a) => {

  return function (){

      console.log("Inside inner Fun")

  }

}

let i = outerFun();

i();

Q.20) What is the advantage of closure?

The main advantage of using closure is that any member which is private for certain scope, can be accessed keeping it private so that the variable is away from any accidental change of value.

Accessing private member with a closure pattern assure better approach of making a variable global.