**Assignment JS**

**Question: 1 to 5**

**Q.1 Explain Hoisting in JavaScript.**

**Ans: Hoisting** is a concept that enables us to extract values of variables and functions even before initializing/assigning value without getting errors and this happens during the 1st phase (memory creation phase) of the Execution Context.

**Features of Hoisting:**

* In JavaScript, Hoisting is the default behavior of moving all the declarations at the top of the scope before code execution. Basically, it gives us an advantage that no matter where functions and variables are declared, they are moved to the top of their scope regardless of whether their scope is global or local.
* It allows us to call functions before even writing them in our code.

**Note:** JavaScript only hoists declarations, not initializations.

JavaScript allocates memory for all variables and functions defined in the program before execution.

**Sequence of variable declaration:** The following is the sequence in which variable declaration and initialization occur.

***Declaration –> Initialisation/Assignment –> Usage***

**Variable lifecycle:**

let a; // Declaration

a = 100; // Assignment

console.log(a); // Usage

**Note:** Always remember that in the background the Javascript is first declaring the variable and then initializing them. It is also good to know that variable declarations are processed before any code is executed.

However, in javascript, undeclared variables do not exist until the code assigning them is executed. Therefore, assigning a value to an undeclared variable implicitly creates it as a global variable when the assignment is executed. This means that all undeclared variables are global variables.

    // Hoisting

**function** codeHoist(){

        a = 10;

        let b = 50;

    }

    codeHoist();

    console.log(a); // 10

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

**Q.2** Explain Temporal Dead Zone?

Ans. The Temporal Dead Zone (TDZ) is a behavior that occurs when trying to access a variable before it has been initialized. When a variable is declared using the let or const keyword, it is hoisted to the top of its scope, but it is not initialized until the line where it was declared is executed. [This means that if you try to access the variable before it has been initialized, you will get a ReferenceError](https://dzone.com/articles/what-is-the-temporal-dead-zone-in-javascript).

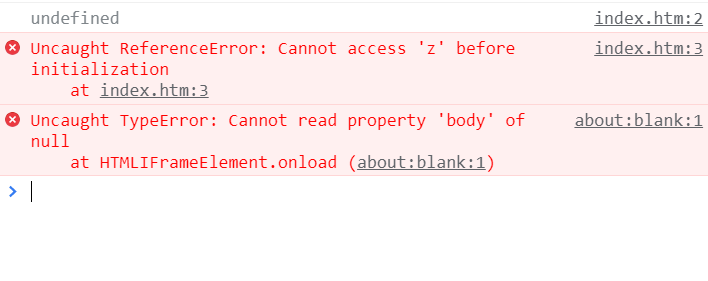
The TDZ is the area of a block where a variable is inaccessible until the moment the computer completely initializes it with a value. A block is a pair of braces ({…}) used to group multiple statements. [Initialization occurs when you assign an initial value to a variable](https://www.freecodecamp.org/news/javascript-temporal-dead-zone-and-hoisting-explained/).

**Example 2:** Now let’s test out the same code snippet by using the let or const keyword.

**index.js**

|  |
| --- |
| console.log(x);      console.log(z);  **var** x = 6;      let z = 6;      console.log(x);      console.log(z); |

**Output:**

**Explanation:**

1. First of all the Global execution context will be created.
2. And then the memory allocation phase starts, During this, the variable x got a place in memory and javascript puts undefined there.
3. And then the variable z gets space in a different place of memory and same as variable x then undefined will be assigned as value.
4. Then the thread execution phase starts, During this console.log(x) statement executes and prints the value of x, which is undefined.
5. In the next line, there is console.log(z), javascript will throw ReferenceError and the program will stop here.

**Q.3** Difference between var & let?

Ans. Var vs. Let Keyword

|  |  |  |
| --- | --- | --- |
| **SN** | **var** | **let** |
| 1. | The var keyword was introduced with JavaScript. | The let keyword was added in ES6 (ES 2015) version of JavaScript. |
| 2. | It has global scope. | It is limited to block scope. |
| 3. | It can be declared globally and can be accessed globally. | It can be declared globally but cannot be accessed globally. |
| 4. | Variable declared with var keyword can be re-declared and updated in the same scope. **Example:**  function varGreeter(){  var a = 10;  var a = 20; //a is replaced  console.log(a);  }  varGreeter(); | Variable declared with let keyword can be updated but not re-declared. **Example:**  function varGreeter(){  let a = 10;  let a = 20; //SyntaxError:  //Identifier 'a' has already been declared  console.log(a);  }  varGreeter(); |
| 5. | It is hoisted. **Example:**  {  console.log(c); // undefined.  //Due to hoisting  var c = 2;  } | It is not hoisted. **Example:**  {  console.log(b); // ReferenceError:  //b is not defined  let b = 3;  } |

**Q.4** What are the major features introduced in ECMAScript 6?

Ans. ECMAScript 2015 was the second major revision to JavaScript.

ECMAScript 2015 is also known as ES6 and ECMAScript 6.

This chapter describes the most important features of ES6.

## **New Features in ES6**

* [The let keyword](https://www.w3schools.com/Js/js_es6.asp#mark_let)
* [The const keyword](https://www.w3schools.com/Js/js_es6.asp#mark_const)
* [Arrow Functions](https://www.w3schools.com/Js/js_es6.asp#mark_arrow)
* [The ... Operator](https://www.w3schools.com/Js/js_es6.asp#mark_spread)
* [For/of](https://www.w3schools.com/Js/js_es6.asp#mark_forof)
* [Map Objects](https://www.w3schools.com/Js/js_es6.asp#mark_map)
* [Set Objects](https://www.w3schools.com/Js/js_es6.asp#mark_set)
* [Classes](https://www.w3schools.com/Js/js_es6.asp#mark_class)
* [Promises](https://www.w3schools.com/Js/js_es6.asp#mark_promise)
* [Symbol](https://www.w3schools.com/Js/js_es6.asp#mark_symbol)
* [Default Parameters](https://www.w3schools.com/Js/js_es6.asp#mark_param)
* [Function Rest Parameter](https://www.w3schools.com/Js/js_es6.asp#mark_rest)
* [String.includes()](https://www.w3schools.com/Js/js_es6.asp#mark_includes)
* [String.startsWith()](https://www.w3schools.com/Js/js_es6.asp#mark_startswith)
* [String.endsWith()](https://www.w3schools.com/Js/js_es6.asp#mark_endswith)
* [Array.from()](https://www.w3schools.com/Js/js_es6.asp#mark_array_from)
* [Array keys()](https://www.w3schools.com/Js/js_es6.asp#mark_array_keys)
* [Array find()](https://www.w3schools.com/Js/js_es6.asp#mark_array_find)
* [Array findIndex()](https://www.w3schools.com/Js/js_es6.asp#mark_array_findIndex)
* [New Math Methods](https://www.w3schools.com/Js/js_es6.asp#mark_math_methods)
* [New Number Properties](https://www.w3schools.com/Js/js_es6.asp#mark_number_properties)
* [New Number Methods](https://www.w3schools.com/Js/js_es6.asp#mark_number_methods)
* [New Global Methods](https://www.w3schools.com/Js/js_es6.asp#mark_global_methods)
* [Object entries](https://www.w3schools.com/Js/js_es6.asp#mark_entries)
* [JavaScript Modules](https://www.w3schools.com/Js/js_es6.asp#mark_modules)

1. **JavaScript let**

The let keyword allows you to declare a variable with block scope.

Example

var x = 10;

// Here x is 10

{

let x = 2;

// Here x is 2

}

// Here x is 10

1. **JavaScript const**

The const keyword allows you to declare a constant (a JavaScript variable with a constant value).

Constants are similar to let variables, except that the value cannot be changed.

Example

var x = 10;

// Here x is 10

{

const x = 2;

// Here x is 2

}

// Here x is 10

1. **Arrow Functions**

Arrow functions allows a short syntax for writing function expressions.

You don't need the function keyword, the return keyword, and the curly brackets

Example

// ES5

var x = function(x, y) {

return x \* y;

}

// ES6

const x = (x, y) => x \* y;

1. **The Spread (...) Operator**

The ... operator expands an iterable (like an array) into more elements:

Example

const q1 = ["Jan", "Feb", "Mar"];

const q2 = ["Apr", "May", "Jun"];

const q3 = ["Jul", "Aug", "Sep"];

const q4 = ["Oct", "Nov", "May"];

const year = [...q1, ...q2, ...q3, ...q4];

The ... operator can be used to expand an iterable into more arguments for function calls:

Example

const numbers = [23,55,21,87,56];

let maxValue = Math.max(...numbers);

1. **The For/Of Loop**

The JavaScript for/of statement loops through the values of an iterable objects.

for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

The for/of loop has the following syntax:

for (variable of iterable) {

// code block to be executed

}

variable - For every iteration the value of the next property is assigned to the variable. Variable can be declared with const, let, or var.

iterable - An object that has iterable properties.

Looping over an Array

Example

const cars = ["BMW", "Volvo", "Mini"];

let text = "";

for (let x of cars) {

text += x + " ";

}

1. **JavaScript Maps**

Being able to use an Object as a key is an important Map feature.

Example

const fruits = new Map([

["apples", 500],

["bananas", 300],

["oranges", 200]

]);

1. **JavaScript Sets**

Example

// Create a Set

const letters = new Set();

// Add some values to the Set

letters.add("a");

letters.add("b");

letters.add("c");

1. **JavaScript Classes**

JavaScript Classes are templates for JavaScript Objects.

Use the keyword class to create a class.

Always add a method named constructor():

Syntax:

class ClassName {

constructor() { ... }

}

Example

class Car {

constructor(name, year) {

this.name = name;

this.year = year;

}

}

The example above creates a class named "Car".

The class has two initial properties: "name" and "year".

A JavaScript class is not an object.

It is a template for JavaScript objects.

1. **JavaScript Promises**

A Promise is a JavaScript object that links "Producing Code" and "Consuming Code".

"Producing Code" can take some time and "Consuming Code" must wait for the result.

Promise Syntax

const myPromise = new Promise(function(myResolve, myReject) {

// "Producing Code" (May take some time)

myResolve(); // when successful

myReject(); // when error

});

// "Consuming Code" (Must wait for a fulfilled Promise).

myPromise.then(

function(value) { /\* code if successful \*/ },

function(error) { /\* code if some error \*/ }

1. **The Symbol Type**

A JavaScript Symbol is a primitive datatype just like Number, String, or Boolean.

It represents a unique "hidden" identifier that no other code can accidentally access.

For instance, if different coders want to add a person.id property to a person object belonging to a third-party code, they could mix each others values.

Using Symbol() to create a unique identifiers, solves this problem:

Example

const person = {

firstName: "John",

lastName: "Doe",

age: 50,

eyeColor: "blue"

};

let id = Symbol('id');

person[id] = 140353;

// Now person[id] = 140353

// but person.id is still undefined

1. **Default Parameter Values**

ES6 allows function parameters to have default values.

Example

function myFunction(x, y = 10) {

// y is 10 if not passed or undefined

return x + y;

}

myFunction(5); // will return 15

1. **Function Rest Parameter**

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

Example

function sum(...args) {

let sum = 0;

for (let arg of args) sum += arg;

return sum;

}

let x = sum(4, 9, 16, 25, 29, 100, 66, 77);

1. **String.includes()**

The includes() method returns true if a string contains a specified value, otherwise false:

Example

let text = "Hello world, welcome to the universe.";

text.includes("world") // Returns true

**Q.5 What is the difference between let and const ?**

**Ans.**

| **let** | **const** |
| --- | --- |
| The scope of a[*let*](https://www.geeksforgeeks.org/javascript-let/) variable is block scope. | The scope of a [*const*](https://www.geeksforgeeks.org/javascript-const/) variable is block scope. |
| It can be updated but cannot be re-declared into the scope. | It cannot be updated or re-declared into the scope. |
| It can be declared without initialization. | It cannot be declared without initialization. |
| It cannot be accessed without initialization otherwise it will give ‘referenceError’. | It cannot be accessed without initialization, as it cannot be declared without initialization. |
| Hoisting is done, but not initialized (this is the reason for the error when we access the let variable before declaration/initialization | Hoisting is done, but not initialized (this is the reason for the error when we access the const variable before declaration/initialization |