**JAVASCRIPT:**

Shortest JS program: Empty file

Java script engine creates this keyword and window.

Window is a global object which is created along with the global execution context. Whenever JS program is running there must be a JS engine. The global object in case of browser is known as window.

The window looks like the below snap and it has several variables and function which can be used anywhere in the code.

Graphical user interface, text, application, email

Description automatically generated

At the global level this == window. Everything you see in top-level is global space. All the variables and function are included in global space, but if a variable is mentioned/declared inside a function that can not be call as a part of global space.

Supoose,

A = 5, similarly, console.log(a)- O/P: 5

Console.log(window.a)

O/P: 5

So, even when we do not mention the window it will automatically point to global space and the same memory phase.

Undefined and not defined: As we know, in the memory creation phase the memory will be reserved for the variable “a” even before it actually assigned its value.

**Undefined** is a special type of keyword that act as a place holder for the variable,

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

**Not defined:** When there is no value of x in the whole program.

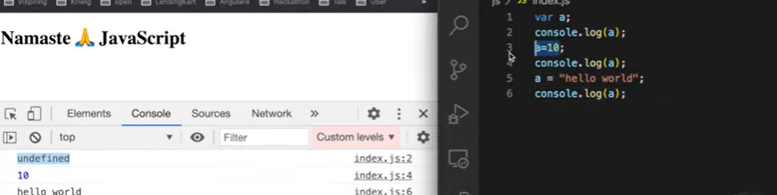
A screenshot of a computer screen

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

JavaScript is a loosely/weakly type language. That meaning if we put a int type datatype in a variable later on we can put string as well.



**Scope chain, scope and lexical environment:**

**Graphical user interface, application

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

Scope: Scope means where you can access a specific variable and function in our code. It is directly dependent on lexical environment.

Two ques: 1) what is the scope of function b ?

2) Can we use b inside of function c?

Lexical environment: Whenever, a global execution context is created a lexical environment is also created. Lexical environment is a local memory along with its lexical environment with its parent. Lexical means hierarchy or in-order, from the above exam we can say the c function is lexically sitting under a function. And a is lexically in global space.

Text

Description automatically generated

From this we can say the orange part denotes the reference of the lexical environment of c, meaning a.

Diagram

Description automatically generated

When you put a console.log then JS engine will try to find that particular variable in the its local space first, if it is not inside the local space meaning inside the same function then it will find that in its lexical memory space. If it was not there also the it will find that other’s function lexical environment also and so on.

A screenshot of a computer

Description automatically generated with low confidence

In this case we will find the value of console.log(b) as 10.

**Scope Chain:**

This way of finding the value inside a code is called scope chain. The whole chain of lexical environment is called scope chain.

Lexical environment of a:

Graphical user interface, text, application, email

Description automatically generated

Lexical environment of c:

Graphical user interface, text, application

Description automatically generated

Diagram

Description automatically generated

Are let and const hoisted ?

Yes, let and const are hoisted in JavaScript. Hoisting means the memory is allocated to variables and constant even before it initializes any value, and you can access it before initialization and it will not give an error.

In the case of var the memory is allocated inside the global space but for let and const the memory is allocated inside the script.

Graphical user interface, text, application

Description automatically generated

**Temporal dead zone** is the time since the let is hoisted till the time it initializes some value in the variable. Whenever you will try to access any variable inside a temporal dead zone it will give you a ReferenceError, it will only be accessed once you initialize value into the variable

**Window object:-** we can access the value of var from window object as the memory is allocated inside the global space but we can not access the value from window object in case of let and const as the memory is allocated in

A picture containing text, device, meter, gauge

Description automatically generated Graphical user interface, text, application

Description automatically generated

In JS, we cannot use the same variable name in the same scope again.

**Const:** we can hoist const also in same way as let but memory will allocates inside script.

But the difference between let and const is that in const the variable value must be initialized.

**Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated**

**Text, application

Description automatically generated with medium confidence**

**Graphical user interface

Description automatically generated with low confidence**

**Difference between TypeError, SyntaxError and ReferenceError:**

1. TypeError means the variable value is already assigned and it is a type of variable. It should be initialized and declared together.
2. SyntaxError means missing syntax. Duplicate declaration is also an syntay error.
3. RefenrenceError means when javascript tries to find a particular variable inside a memory space and it cannot access it.

**Let, Const, var which type of variable we should use more?**

We should always use Const when we know the value of the variable will not be changed. If not const then use let if possible to avoid unexpected errors.

**Blockscope and Shadowing in JavaScript:**

**Block:** The curly braces are called a block inside a code. Block is also known as a compound statement. Block means where a group of statements can be used in a space where JS expects only one statement. The statement will execute one after another inside a block.

**Block Scope:** The variables and function you can use inside a block is called block scope. So, from the below picture we can say that let and const are block scoped as their memory is stored inside a block scope where the memory for var is stored in global scope.

**Text

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated**

**Shadowing in Js:**

If two variable has the same name then the 2nd variable will shadow the first variable and will also modify the value of it.

Suppose we have, var a = 100;

Then second line, var a =10;

So, a will always take the value of 10 in case of var.

Text

Description automatically generated Graphical user interface, text, application

Description automatically generated

Here, we have three types of scope.

Shadowing not only works inside a block but also works the same way inside a function. In the case of let and const, it will only shadow the value inside a block/function outside of block/function it will take the first value. But, in the case of var it will shadow throughout.

**Illegal Shadowing:** If we try to shadow the same variable but a different datatype then it is called illegal shadowing. E.g- You can not shadow a let using a var but vice-versa is possible here.

**Closure in JS :**

A closure is the combination of a function bundled together with the reference of its lexical environment.

**Text

Description automatically generated**

In this picture, the z will return the value 7 as it will not only give its scope but also its lexical environment.

Uses of closure:

1. Module design pattern
2. Currying
3. memoize
4. Function like once
5. Maintaining state in async world
6. setTimeouts
7. Iterators