HOISTING:

Same types of variables are accessible /useable in the code before their declaration.

VARIABLES lifted to the top of their scope.

(BEHIND THE SCENES): Before execution, code is scanned for the variable declarations and for each variable, a new property is created in the variable environment object.

|  |  |  |  |
| --- | --- | --- | --- |
|  | HOISTED | INITIAL VALUE | SCOPE |
| FUNCTION DECELARATION | YES | ACTUAL FUNCTION | BLOCK |
| VAR | YES | UNDEFINED | FUNCTION |
| LET | NO | UNINITILIAZED(TDZ) | BLOCK |
| FUNCTION EXPRESSION AND ARRAY | DEPENDS IF USING VAR OR LET/CONST |  |  |

**Let and const are not actually hoisted:**

Technically, there are hoisted but their value is basically set to uninitialized.

In practice, it is as if hoisting is not happening at all.

Rather, we say those variables are placed in so called.

**TERMINAL DEAD ZONE:** which doesn’t let us access the variables between the beginning of the scope and the place where the variables are declared.

Example:

‘use strict’

Let start = “board” + finish;

Console.log(start);

Const finish = context;

**Why TDZ??(comes with ES6 🡸 ECMA script6{JS version})**

* Makes it easier to avoid and catch errors. -->accessing variables before declaration is a bad practice and it should be avoided.
* Makes const variables actually work.🡪As we can’t reassign the const variables so it will be possible to set them undefined first and then assign real value problems.

**‘this’ keyword:**

* A special variable that is created for every execution context (every function).
* Takes the value of (points to) the “owner” of the function in which this keyword is used.
* ‘this’ is not static: It depends on how the function us called the value is only assigned when the function is actually called.

***Analysis of function calling in different ways***.

* First way to call a function is as a method so as a function attached to an object.
* Method: this =<object that is calling the method>-🡪 when we call a method then this keyword inside that method will simply point to object on which the method is called.
* This=>A special variable that is created for every execution context (every function) takes the value of (points to) the “owner” of the function in which this keyword is used.

Example:

const summer = {

name : “vinay”

year = 2003

college : function(){

return “2020-2024”;

}

}

2.Simple function call:

This=> undefined

**OOPs:**

Object oriented programming basically oops is a programming paradigm based on the concepts of classes and objects.

1. We use objects to model (describe)real world entities or abstracted features.
2. Objects may contain data(properties) and code (method). By using objects, we pack data and the corresponding behaviour into one block.
3. In OOPs, objects are self-contained area/block of code.
4. Objects are building blocks of applications and interact with one-another.
5. Interaction happens through a public interface method that the code outside of the object can access and use to communicate with the object.
6. OOPs was developed with the goal of organizing code to make it more flexible and easier maintenance.