Rahul Aher

Page No. Date

(1)

Javascript - Day 3

* What is HOISTING ?

* Example 1 :-

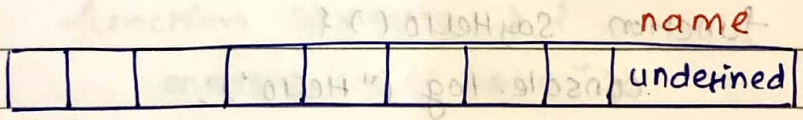
```
console.log (name);
var name ="Rahul Aher";
```

Output >> undefined.

→ Why undefined & shouldn't it say name doesn't exist (Reference Error).

It's because of HOISTING !!!

→ During the creation phase JE sees this variable (name) and allocates memory for it and keeps undefined as it's value.

Memory 

→ So, since hoisting happens in creation phase there is already a variable name name in the memory with value undefind.

So during execution phase when it runs the file from top to bottom this is what it sees.
[not physically well sees that in a while]

```
var name = undefined ;
console.log (name);
name = 'Rahul Aher';
```

$\xrightarrow{}$ this is hoisting

note this is an assignment and does not have var.

NOTE : variables are partially hoisted.

Meaning :→ They are allocated memory but not assigned actual value what we give.
→ They are simply assigned undefined.

* Example 2 :-

```
var name = "Rahul Aher";
var name = "Rahul Aher";
```

● What about this? How many times will name be hoisted?

→ ONCE...!!! Why? since the value doesn't matter, JE will say. I already, have hoisted name.

* Example 3 :-

```
sayHello ();
function sayHello () {
    console. log ("Hello");
}
```

output >> Hello

→ Just like variables, functions are also hoisted. But !!! functions are completely hoisted. meaning, The entire function with defination is hoisted. So, it won't gives us undefined.

JUST FOR CLARIFICATION :-

JE doesn't physically moves variables and functions to the top of file. It just allocated memory for them before running the code.

-Rahul Aher

Page No. ③

* Example 4 :-

```
sayHello ();
var sayHello = function () {
    console.log ("Hello");
}
```

output >> undefined.

it will throw error saying that sayHello is not a function.

→ It was treated like a var and given undefined as it doesn't care about that value.

* Example 5 :-

```
sayHello ();
function sayHello () {
    console.log ("Hello");
}
function sayHello () {
    console.log ("Bollo");
}
```

output >> Bollo

→ Since functions are completely hoisted, when same function comes 2nd time, JE will say I have already allocated memory for sayHello() let me replace it's content from console.log("Hello"); to console.log("Bollo");

Rahul Aher

Page No. ④
Date

* Example 6

TRICKY ONE

Just remember when a function is called a new execution context for that function get's created.

```
var name = "Rahul Aher";
var changeName = function (){
          console.log ("Name", name);
          var name = "John";
          console.log ("changed Name", name);
};

changeName ();
```

* What you probably expect?

```
> Name Rahul Aher
> changed Name John
```

✗ Wrong

```
> Name undefined
> changed Name John
```

✓ Correct

* What?? & Why??

→ When you called a new function execution context was created on the top of global execution context.

→ This execution context is created for the function when it is called.

name: undefined
(since we ~~defined~~ declared name again in func)

← Execution context of sayHello()

sayHello : fn()
name : undefined

← global Execution context.

→ Go an look at the problem again, you will get it. :)

→ BUT still, JS is tricky and weired

→ Oh! guess why they introduced let & const :)
They are Hoisted ....!!!

→ Even though they are hoisted they are not assiged anything (not even undefined) and we ger ref. error if we try to access them before using ~~ten~~ them.