

# What is hoisting?

Ex 1

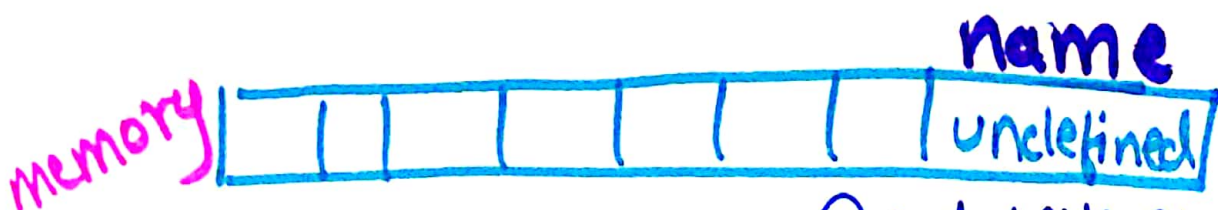
@codeWithSimran

```
console.log(name)
var name = 'Simran'
> undefined
```

Why undefined? Shouldn't it say name ~~is~~ ~~no~~ 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 its  
value.



@codeWithSimran

~~E~~ So since hoisting happens in creation phase there is already a variable named name in the memory with value undefined. @codeWithSimran

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

var name = undefined

console.log(name)

name = 'Simran'

this is hoisting

note this is an assignment and does not have var

@codeWithSimran



NOTE: Variables are partially hoisted

@codeWithSimran

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

Example 2

```
var name = 'Simran'
```

```
var name = 'Simran'
```

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

ONCE!! Why? Since the value doesn't matter, JS will ~~the~~ say I already have hoisted name.

### Example 3

@codewithSimran

```
sayHello();
```

```
function sayHello() {  
  console.log('say Hello')  
}
```

```
> Hello
```

Just like variables, functions are also hoisted.

BUT!! Functions are completely hoisted.

Meaning: The entire function with definition is hoisted.

So it won't give us

undefined. @codewithSimran



## Example 4

@codeWithSimran

### Just a clarification

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

sayHello(); function expression

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

> undefined

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

## Example 5

SayHello();

@codeWithSimran

~~SayHello~~

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

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

> Bollo

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



## Example 6

@codeWithSimran

### TRICKY ONE

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

```
var name = 'Simran'
```

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

@codeWithSimran

```
changeName();
```

what you probably expect

> Name Simran

> Changed Name John

Wrong  
⊗

> Name undefined

> changed Name John



What?? Why?

@codewithSimran

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

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

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

← execution context of sayHello()

sayHello: func  
name: undefined

→ global execution context



Go and look at the problem again, you'll get it. :)

BUT STILL, JS is tricky and weird :/

Oh! guess why they introduced let and const :)

They are ~~not~~ hoisted !!!

@codeWithSimran

Even though they're hoisted they're not assigned anything (not even undefined) and we get reference error if we try to access them before using them