[it is a phenomena by which you can access variables and functions even before they have initialised]

episode 03

Hoisting in Javascript (variables and functions)

-
```
var x = 7;
function getName(){
    console.log("Namaste");
}
getName();
console.log(x);
```

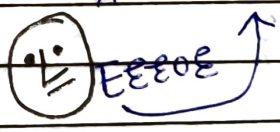
⟹ Namaste
7

-
```
getName();
console.log(x);
var x=7;
function getName(){
    console.log("Namaste");
}
```

⟹ Namaste
undefined

-
```
getName();
console.log(x);
function getName(){
    console.log("Namaste");
}
```

⟹ Namaste
"x is not defined"



-
```
var x=7;
function getName(){
    console.log("Namaste");
}
console.log(getName);
```

⟹
```
f getName(){
console.log("Namaste");
}
```

[Here it printed the function itself]

[it allocates memory in phase I]
[this is due to execution context]

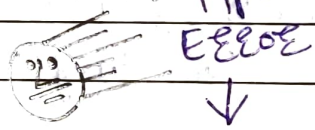
Page No.
Date

[will give function]

-
```
console.log(getName);
var x=7;
function getName(){
    console.log("Namaste");
}
```
⟹
```
f getName(){
console.log("Namaste");
}
```

-
```
getName();
console.log(x);
console.log(getName);
var x=7;
function getName(){
    console.log("Namaste");
}
```
Namaste
undefined
⟹
```
f getName(){
console.log("Namaste");
}
```

[only like this, it is hoisted]

-
```
getName();
console.log(x);
console.log(getName);
var x=7;
var getName = () => {
    console.log("Namaste");
}
```


⟹ getName is not a function

undefined ←

[we have used arrow function here]

[arrow function will be stored as undefined just like variable]

- Conceptually, for example, a strict definition of hoisting suggests that variable and function declarations are physically moved to top of your code, but this is not in fact what happens. Instead, the variable and function declarations are put into memory during compile phase, but stay exactly where you typed them in your code.

=> To avoid bugs, always declare all variables at the beginning of every scope

Page No.
Date

- one of the advantages of javascript functions declarations into memory before it executes any code segment is that it allows you to use function before you declare it in your code.

- we can call our function in our code first, before the function is written, the code will still work. This is because of how context execution works in javascript.

- (*) only declarations are hoisted, not initializations in case of variables.

if a variable is declared and initialized after using it, the value will be undefined.

ex.

```
- console.log(num);          ⟹ undefined
  var num;    //declaration
  num = 6;    // initialization
```

- The following example has only initialisation. No hoisting happens so trying to read the variable results in "Reference Error" exception.

```
console.log(num);   // ReferenceError exception
num = 6;            // initialisation
```

[cae = "hello"; let cae; → ReferenceError]

[cae = "hello"; const cae; → SyntaxError (code will not run)]

Summary :- Hoisting as a core concept relies on the way how execution context is created. In the first case/phase i.e. memory allocation phase all the variables and functions are allocated memory, even before any code is executed. All variables are assigned undefined at this point in time in local memory.

- Initialisations using 'let' and 'const' are also not hoisted.