Hoisting :

Hoisting is a phenomenon by which we can access the variables & functions even before initialization.

Ex:

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

O/P:

```
Hello world
undefined
ƒ getName() {
    console.log("Hello world");
}
```

Ex:

```
getName2();
getName();
console.log(getName);
console.log(x);

x=7;
var getName = () => {
    console.log("Helloworld");
}
var getName2 = function() {
    console.log("Hello world");
}
```

o/p :

~~getName2 function is not defined~~

~~function is not defined~~

getName2() is not a function

Ex:

```
getName2();
getName();
console.log(getName);
console.log(x);

x = 7;

function getName() {
    console.log("Hello Javascript");
}

function getName2() {
    console.log("Hello Javascript");
}
```

o/p:

```
Hello Javascript
Hello Javascript
f function: getName()
```

ReferenceError: x is not defined

SCOPE :-

-> Javascript has 3 types of scope

- Block scope
- Function scope / Local scope
- Global scope

Block scope :

-> Before ES6, JS variables had only Global scope and Local scope

-> ES6 introduces two important new keywords: let & const.

-> variables declared inside a { } block cannot be accessed from outside the block

Eg:

```
{
    let a = 2
}
// a cannot be used here
```

Local Scope :

-> variables declared within a Javascript function, are Local to the function.

-> Local variables can only be accessed within the function.

Ex:

```
// code here can not use carName

function myFunction() {
    let carName = "volvo";
    // code here can use carName
}

// code here cannot use carName
```

Global scope :

-> variables declared aside a function, becomes "Global".

-> Global functions and all scripts can use Global variables.

Ex:

```
let carName = "volvo";
// code here can use carName
function myFunction() {
    // code here can also use carName
}
```

Ex:

```
var x = 1;                          // Global scope

a();
b();
console.log(x);

function a(){
    var x = 10;                     // Local scope
    console.log(x);
}
function b(){
    var x = 100;
    console.log(x);
}
if (true) {
    let x = 1000;                   // Block scope
    console.log(x);
}
console.log(x);
```

o/p:

```
10
100
1
1000
1
```

Undefined Vs Not-Defined :

→ "undefined is a special placeholder which is used to reserve memory for the variables in the memory creation phase. Even before a single line of code is executed, JS engine assigns undefined to the variable.

→ Not-defined means if we try to console or access any variable which is not declared in the code then we get not-defined error.

→ undefined != not defined

becoz undefined means variable is declared but no value is assigned but not defined means variable is not even declared.

→ JS is a loosely typed language means it does not attaches its variables to specific data types like other programming languages (c++, Java)