execution context a.k.a scope

definition:

📌 stack-frame which contains args & local variables to execute set of instructions

2 phases

- phase -1 : creation / push

📌 variables which are declared with 'var' keyword,
 always will get hoisted to top of the scope with default value i.e undefined

- phase -2 : execution / pop

📌 by default , javascript runtime has one global-scope

📌 every function call creates new-scope, which is child scope of on which scope that function has declared/created

---

Quiz-1

```
debugger
console.log("-first-line-")
console.log(v)
```

Quiz-2

```
debugger
var v = 12
function f1() {
    console.log(v)
    var v = 13
}

f1() // f1-scope <= global-scope
```

Quiz-3

```
debugger;
var v=12  // global scope
function f1(){
    function f2(){
        console.log(v)
    }
    f2(); // f2-scope <=== f1-scope
    let v=13  // f1-scope
}
f1(); // f1-scope <=== global-scope
```

Quiz-4

```
var v = 12
var v = 13
```

Quiz-5

```
var v = 12
if (true) {
    var v = 13  // by default no block-scope, we have function scope
}
console.log(v)
```

problems with '**var**' keyword based variables

- variable get hoist
- can re-declare same variable within scope
- no block-scope

solution:

📌 using '**let**' & '**const**' keywords

```
//-----------------------------------------------

console.log(v) // error
let v=12

//-----------------------------------------------

let v = 12
let v = 13 // error

//-----------------------------------------------

let v = 12
if (true) {
    let v = 13 //  block-scoped
}
console.log(v)

//-----------------------------------------------
const person = {
    name: 'Nag'
}

// person=null // error
person.name = "indu" // properties are mutable

//-----------------------------------------------
```

---

summary:

- use 'let' keyword for mutable reference
- use 'const' keyword for immutable reference
- avoid 'var' keyword for better/error-free code

📌 is it possible to use **var** in nested loop because there is no block scope?
