Code block { ... }

Nested Function

```
Function Func1 () {
  let KaK = " --- "
  Function Func2 () {
  }
}
```

Can be returned as a result or as a Property of a new object → still has access to the outer Variables

Lexical Environment

1. Variables

every running Function, Code block { --- }, and the script as a whole have an internal (hidden) associated object Known as (lexical env.)

lexical env.
- → env. record ⇒ object stores all local Var. as its Proper. [and some other info like this]
- → refrence ⇒ to the outer lexcial env.

```
let Phrase = "hello";
alert(Phrase)
```

[Phrase: <uninitialized>] → [Phrase: "hello"] → (null) → global معندوش outer

global lexc. env. فبيعتبر مش جوا code block

lex. env. → is a specification object "theoritical"

2. Function

A Func. is a Value like Var. but it is instantly Fully intialized تقدر تستخدمها من قبل ما توصل لسطر declar. على خلاف الـ Var.

```
let phrase = "hello";
Function Say (name) { }
Say ("John")
```

[name: John] → outer → [Say: Function, Phrase: "hello"] → null

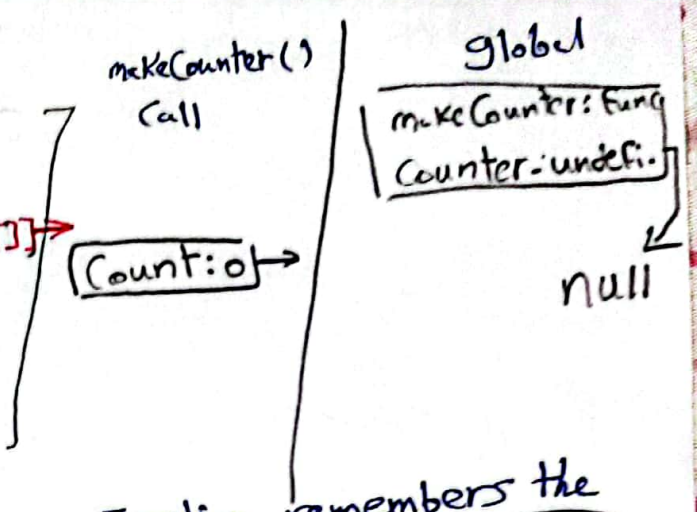
(earlier: [Phrase: <uninitialized>, Say: Function])

inner | outer

when the Code wants to access a Variable search inner lex. env. → search outer lex. → search outer → ... if not Found in strict mode ⇒ error

3. Returning a Function

```
Function makeCounter() {
    let count = 0;
    return Function() {[[Environment]]
        return count++;
    };
}
let counter = makeCounter();
```



Function remembers the lex. env. in which it was made → hidden Property named [[Environment]]

Closure: Function that rem. its outer var. and can access them [in JS All Func. are closures]

Garbage Collection

lexC. env. is removed from memory after the Function call Finishes (it is not reachable)

But if there are nested Functions the lex-env. exists only while there's (at least one) nested Function refrencing it.