Start the time and move ahead

Closures

```
1  function todo(task) {
2      setTimeout(function fun() {
3          console.log("Completed", task);
4      }, 2000);
5  }
6
7  console.log("Starting");
8  todo("assignments");
9  console.log("Ending");
```

→ calling todo

→ At the time instance when function fun will be called, the todo function is already over

→ Starting
Ending

if fun is called after the completion of todo, & todo has ended, how fun is able to access task variable

This happens due to closures.

Closure is when a function "remembers" it's lexical scope even when the function is executed outside that lexical scope.

if todo is completed it will be removed from callstack. how actually the variables still persist ?? where are they ??

```
function fun(task1 , taks2){
    task1 = "paras"
    setTimeout(function gun(){
      console.log("completed" , task1);

    }, 2000)
    task1 = task2 ;
    task2 = "assign";
}

fun("12", "34");
```

task1 = "34"

task2 = "assign"

```
function test() {
    for(var i = 0; i < 3; i++) {
        setTimeout(function exec() {
            console.log(`i : ${i}`); // 'i: ' + i
        }, i*1000);
    }
}

test();
```

func^ scope → test

i = ∅ X ∉ 3

Time → 0ms

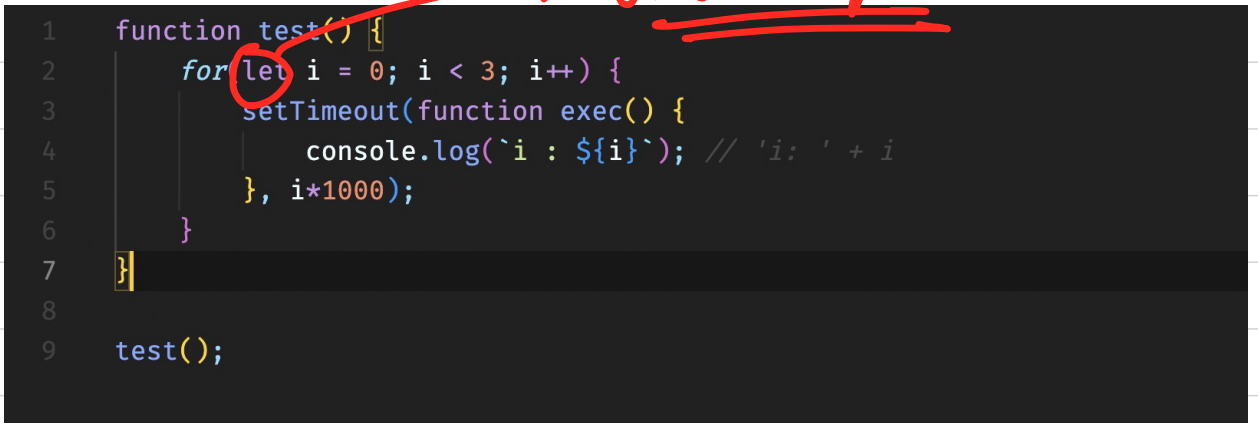
Time → 1s

Time → 2s

i : 3

i : 3

i : 3



Block scope

i = 0

Time → 0ms

i = 1

Time → 1s

i = 2

Time → 2s

```
function todo(task) {
    setTimeout(function fun() {
        console.log("Completed", task);
    }, 2000);
}

console.log("Starting");
todo("assignments");
console.log("Ending");
```