• github : divyaranjan10

DATE

CHAP 11 : setTimeOut + Closures Interview Question

```
function x() {
    var i = 1;
    setTimeOut (function () {
        console.log(i);
    }, 3000);
    console.log ("Namaste JS");
}
x();
```

• Output
Namaste JS
1

• " Time, Tide and Javascript waits for none"
=> So, it takes the callback function and attaches the timer to it and stores it somewhere else, the first it prints "Namaste JS", when the 3000 ms timer expires, it takes the function and puts it again in the call stack and then 1 is printed.

DATE

- Tricky Interview Question:

```
function x() {
    for(var i=1; i<=3; i++) {
        setTimeout(function() {
            console.log(i);
        }, i * 1000);
    }
    console.log("Namaste JS")
}
x();
```

(11.1)

- Output

```
Namaste JS
4
4
4
```

=> So, first 'Namaste JS' will be printed then in (11.1), what happens is the for loop will loop on running and it will ignore 'setTimeOut' as it is asynchronous, it will wait for other tasks to get finished, so basically the function remembers the reference to 'i', the last time the JS engine encountered the reference of 'i', it was 3, therefore 4 is printed 3 times.

DATE

- Solution to the problem:

⇒ Using 'let' instead of 'var' as let is 'block-scoped' whereas 'var' is 'f" scoped', as everytime the loop runs the 'i' is a new variable altogether, therefore each time the 'setTimeout' func' it has a new copy of 'i' with it.

- Extension of Interview question (Solve using 'var' only):

SOLUTION

```
function x() {
  for(var i=1; i<=3; i++){
    function close(i) {
      setTimeout(function() {
        console.log(i);
      }, i * 1000);
    }
    close(i);
  }
  console.log("Namaste JS");
}
x();
```

- Output

Namaste JS
1
2
3

DATE

=> So, to get the solution, everytime we have to provide with the new (updated) copy of 'i', and it we can achieve it using closure. (we can form a closure). Therefore, we'll create a function close() and everytime we invoke it, we will provide with the new copy of 'i'.