13 video

SetTimeout + closures

```
function x()
{ var i = 1;
  setTimeout( function() {
      console.log(i);
  }, 3000);
}
x();
```

O/P

Namaste JS

1 ← After 3s

Time in milliseconds

It will log 1 after 3 seconds

Explanation

setTimeout take callback function and stores it some where & sets timer of given time. And Rest do not wait. It keeps executing. When time over, setTime function/variable also logged on console.

JS do not wait for anything.

Print 1 after 1s, 2 after 2s & so on till 5

```
function x() {
  for( var i = 1; i <= 5; i++) {
    setTimeout( function() {
      console.log(i);
    }, i * 1000);
  }
  console.log("Namaste JS");
}
x();
```

Wrong co

JS do not behaves in t way.

O/P

Namaste JS

6 → after 1s

" " 2s

" " 3s

" " 4s

JS working in this due to closure

Closure

→ variables bundled with closures points to some memory location i.e. their reference is bundled.

→ JS do not waits for anything.

→ when loop breaks values of i will be 6. it

→ setTimeout puts function in a separate place.

→ JS runs for loop 5 times and each setTimeout put separate copy function but variable pointing to some memory location (reference).

When time overs, it prints value pointed by variable i [6].

```
→ function x() {
    for(let i=1; i<=5; i++) {
      setTimeout(function(){
        console.log(i);
      }, i*1000);
    }
    console.log("Namaste Javascript");
  }
  x();
```

O/P

Namaste Javascript
1 → After 1sec
2 " " 2sec
3 " " 3sec
4 " " 4sec
5 " " 5sec

let variables are block scoped for each iteration a completely differently variable let variable will be used having same name Each time a separate closure will be formed will new variable each time referring to different memory allocation.

Camlin Page
Date / /

Without use of let variable

```
function x(){
for(var i=1; i<=5; i++){
  function close(j){
    setTimeout(function(){
      console.log(j);
    }, j*1000);
  }
  close(i);
}
console.log("Namaste JS");
}
x();
```

value of i will be copied into j and each new j variable will used

O/P

Namaste JS
1 → 1sec
2 → 2sec
3 → 3sec
4 → 4sec
5 → 5sec