Page No.
Date

* Episode 14 *

* Callback Functions

We know that functions are first class citizens in javascript. i.e. we can take a function and pass it into another function and when you do so, the function which you pass into another function is called as callback function.
These callback functions are very powerful in javascript, it gives us access to the whole asynchronous world in a synchronous single threaded language.

single threaded synchronous means it can do one thing at a time and in a specific order.

But due to callback we can do async things inside javascript.

example :-

```
function x() {
    // code
}
x(function y() {
    //code
});
```

→ if you call a function and if you pass a function inside another function, these function y is called as callback function

Page No.
Date

Now let us see, how these callback function is used in asynchronous task.

example :-

```
setTimeout(function () {
    console.log("timer");
}, 5000);
function x(y){
   console.log("x");
}
x(function y(){
   console.log("y");
});
```

x
y
timer

As javascript is single threaded language, code will execute one line at a time in specific order. so first thing is happen is, registering a setTimeout so setTimeout will take a callback function and store it in a seperate space and will attach a timer of 5000 milliseconds.
As we already know, javascript won't wait for setTimeout to finish, that is why we say callback function gives us power of asynchronity, it does not wait for 5000 milli seconds.
[javascript waits for none]

so now program will move on to next part of code.
it will see a function definition of x and then it will try to call x function. It will pass y i.e. callback function into y and will execute code.
so first it will print console.log("x") and then console.log("y")
After some time 5000 millisecond expires, and that expires then callback function is executed.

so setTimeout asynchronous operation was not possible without callback.

* Blocking Main Thread in Javascript

Javascript has just one call stack and we can also call it as main thread.
so whatever is executed inside your page is executed through call stack only. so if any operation blocks the call stack, that is called as blocking the main thread.
we should never block our main thread, instead we should use async operations for things which takes time.

* example of closure with event listener

```
function attachEventListeners(){
    let count = 0;
    document.getElementById("clickMe")
    .addEventListener("click", function xyz(){
        console.log("Button clicked", ++count);
    });
};
attachEventListeners();
```

↓

whenever user clicks on button
Button clicked 1
Button clicked 2
so on

The reason why we have declared count variable inside function is because, declaring it in global space is not a good practice. And when we declare it inside function then we get advantage of data hiding.

function xyz is callback function, whenever user clicks on button, this xyz function will be executed.

And whenever user is clicking on button, callback function increases the count, this is due to closure.

interview question

IMP

* Garbage collection and remove event listeners

→ Why do we need to remove event listeners

first of all, event listeners are heavy i.e. it takes more memory.
so whenever you attach any event listener, it kind of forms a closure. And even when the call stack is empty but still our code will not free the memory because of event listeners attached.
so in this case we cannot free up these extra memory and that is why event listeners are heavy.
And this is the reason, why we remove event listeners when we are not using them.

so when we remove the event listener then all the variables which were held by closure will be garbage collected.