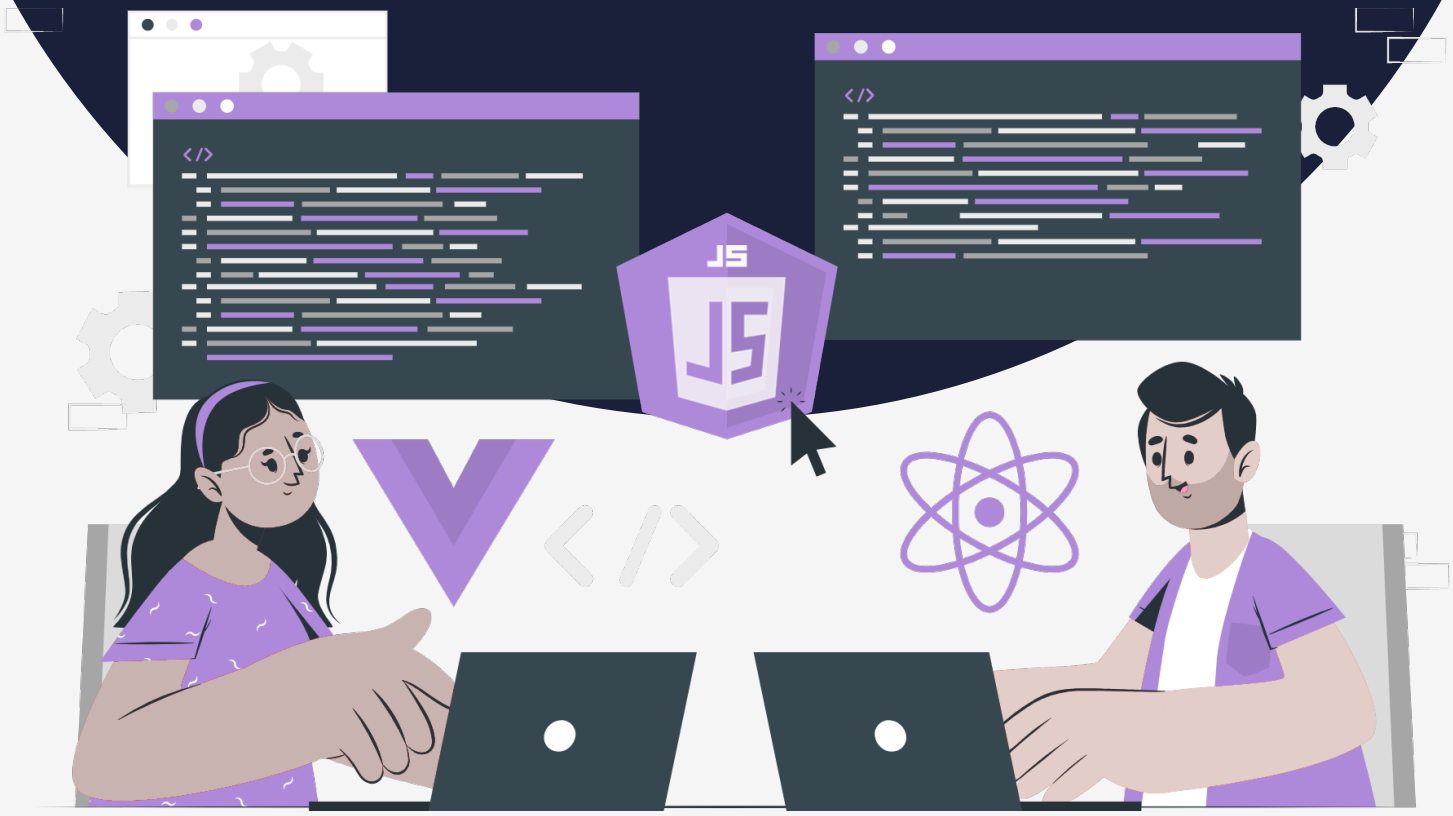
Lesson:

Setting Time, SetInterval and Settimeout



List of content :

By adding specified milliseconds to the date January 1, 1970, the JavaScript Date setTime() Function can be used to obtain a date object.

It takes milliseconds as the parameter and returns an output date by adding these milliseconds to January 1, 1970

The setTime() function helps to assign a date and time to another Date object.

Let us see an example to understand it better:

```
const currentDate = new Date('January 24, 2023, 12:00:00');
const sameDate = new Date();
sameDate.setTime(currentDate.getTime());
console.log(sameDate);
```

Here, we are initializing the currentDate with the value of the current date.
We have created sameDate to show how we can use setTime() to get the same date as out currentDate.

For this, we will have to take help from getTime().
getTime() function helps to extract the exact number of milliseconds since January 1, 1970 00:00:00. This helps us to extract the exact same time for sameDate()

setInterval()
This method repeats a given function at every given time interval and returns an interval ID that uniquely identifies the interval.

Syntax:
```
setInterval(function, delay milliseconds);
```

function: the first parameter is the function to be executed delay milliseconds: indicates the length of the time interval between each execution.

A function to be executed every delay milliseconds. The first execution happens after a delay of the specified milliseconds.
There is also an option of passing arguments to the callback function in setInterval, for that the syntax would be:

setInterval(function,delay milliseconds, arg1, arg2.....argN)

As you can see, you may pass as many arguments as required.
Let us look at a few examples for a better understanding:

Example 1: Displaying messages with the help of setInterval()

```
const exSetInterval = setInterval(myRepeatedMessage, 300);
function myRepeatedMessage()
{
 console.log('Hi');
 console.log('Hi Again !');
}
```

Here, the function myRepeatedMessage() will keep on printing the messages 'Hi' and 'Hi Again' after every 300 milliseconds.

Example 2: Modify the above example by passing arguments to the function.

```
const exSetInterval = setInterval(myRepeatedMessage, 300, 'Hi', 'Hi Again');
function myRepeatedMessage(a,b)
{
 console.log(a);
 console.log(b);
}
```

Example 3: Displaying time with the help of setInterval and another helping function.

```
setInterval(timer, 1000);
function timer() {
  const date = new Date();
  const newTime = date.toLocaleTimeString();
  console.log(newTime);
}
```

For your understanding, the toLocaleTimeString() method returns a string with a language-sensitive representation of the time portion of the date.

clearInterval()

The clearInterval() method cancels a timed, repeating action which was previously established by a call to setInterval().
If the parameter does not identify a previously established action, this method does nothing.

Syntax :
clearInterval(intervalID)

intervalID
The identifier of the repeated action you want to cancel. This ID was returned by the corresponding call to setInterval().

Example 4: How clearInterval works on setInterval()

```
var interval = setInterval(warning, 3000);
 function warning() {
  console.log('3 second warning');
  clearInterval(interval);
}
```

setTimeout()

The setTimeout() method sets a timer that executes a function or specified piece of code once the timer expires.

Syntax:
setTimeout(function,delay,argument1, argument2.....argumentN)

function: is executed after timer expires

delay: in milliseconds is the time for which the timer should wait before the specified function or code is executed. If this parameter is omitted, a value of 0 is used, meaning execute immediately.

arguments: Additional arguments which can be passed through to the function specified here.

The returned timeoutID is a positive integer value which is an identifier for the timer created by the call to setTimeout. We can use this identifier to cancel the timer using clearTimeout.

setTimeout() is an asynchronous function which simply means that the timer function will not pause the execution of other functions while the timer is running.

Example 1 : Display a set of messages using setTimeout()

```
setTimeout(() => {console.log("First message")}, 1000);
setTimeout(() => {console.log("Second message")}, 2000);
setTimeout(() => {console.log("Third message")}, 3000);
```

Output:

```
[Running] node "c:\Users\ACER
First message
Second message
Third message

[Done] exited with code=0 in
```

clearTimeout()

The clearTimeout() method cancels a timeout previously established by calling setTimeout().

Syntax:
clearTimeout(timeoutID)

The clearTimeout() method requires the id returned by setTimeout() to know which setTimeout() method to cancel.

Example 2 : To demonstrate the functioning of clearTimeout()

```
const timeoutId = setTimeout(function(){
    console.log("Hi");
}, 2000);

clearTimeout(timeoutId);
console.log(`Timeout ID ${timeoutId} has been cleared`);
```

Output: