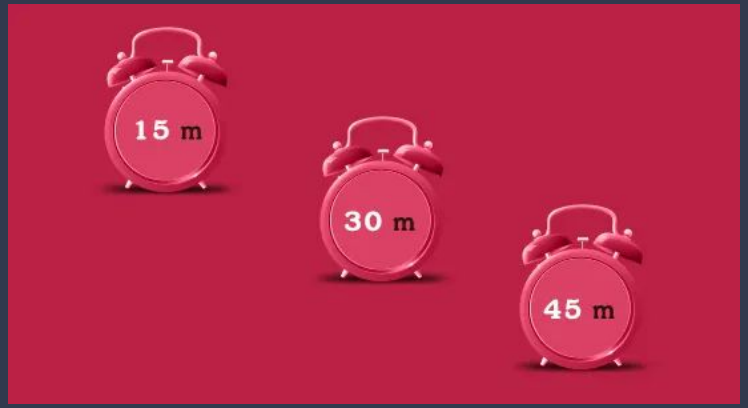
Session 3.0

Timers

Timers

Timers



We may decide to execute a function not right now, but at a certain time later. That's called "scheduling a call".

There are two methods for it:

- setTimeout allows us to run a function once after the interval of time.
- setInterval allows us to run a function repeatedly, starting after the interval of time, then repeating continuously at that interval.

These methods are not a part of JavaScript specification. But most environments have the internal scheduler and provide these methods. In particular, they are supported in all browsers and Node.js.

setTimeOut

setTimeout()

Sometimes it is necessary to program a function call in a specific moment. The JavaScript setTimeout() function is ready to help you with this **task-scheduling** by postponing the execution of a function until the **specified time**.

Let's say you are a developer who wants to enhance user experience in their website. One of the possible features to include is to **schedule a message** to be displayed after users scroll down to the end of the page. JavaScript setTimeout() function can help you set timeout for the method and impress your website visitors with a new feature.

Please bear in mind that the setTimeout JavaScript function will execute functions **once**. If you want the function to be repeated multiple times, you should use setInterval().

setTimeout Syntax

As you can see in the snippet below, the JavaScript setTimeout() function can contain several parameters:

```
setTimeout(function, milliseconds, param_one, param_two, ...)
```

First of all, the setTimeout JavaScript method should contain the **function** that you intend to apply.

The second parameter sets a **time** when the specified function is to be called. However, it is optional, and the default value is 0.

You can also specify **parameters** for the function you have indicated previously.

```
var timer;

const timeOut= () => {
  timer = setTimeout(alertFunc, 2000);
}

const alertFunc = () =>{
  console.log(`Two seconds have passed!`);
}

timeOut();
```

The first example illustrates how JavaScript setTimeout() function can open an alert box after 2 seconds:

```
var timer;

const timeOut = () => {
  timer = setTimeout(() => {
    console.log("This timer will be stopped")
  }, 3000);
}

const stopTimer = () => {
    clearTimeout(timer);
}
```

The last example we prepared will stop the timeout if the `clearTimeout` is called before the timer runs out:

setInterval

setInterval()

- JavaScript interval to be set use setInterval() function. It could be defined simply as a method which allows you to invoke functions in set intervals.
- The JS setInterval() function is similar to the setTimeout method. How are they different? Well, setTimeout() calls functions **once**. However, with the set interval method you can invoke them **multiple** times.
- Let's say you need to display a message to your website visitors every 3 seconds. By applying the JavaScript setInterval() function, you will be able to perform this task and incorporate a new feature on your website.

setInterval Syntax

```
setInterval(function, milliseconds, param_one, param_two, ...)
```

- As you can see, the JavaScript `setInterval()` can contain three parameters. The first one identifies which **function** is going to be applied in the set time intervals. As you might expect, you also should include the **milliseconds** to set the frequency of your function. Additionally, you can specify **parameters** for the function to be applied.

```
var timer;

const interval = () => {
  timer = setInterval(every2sec, 2000);
}

const every2sec = () => {
  console.log("Alert Text!");
}

interval();
```

The example below log alert every 2 seconds:

clearInterval

- In some cases, you might need to make JavaScript stop `setInterval()` from being executed before the times comes. You'll need to use the `clearInterval()` method. It's meant to stop the timer set by using the setInterval JavaScript function.
- The `setInterval()` returns a variable called an interval ID. You can then use it to call the `clearInterval()` function, as it's required by the syntax: `clearInterval(intervalId)`
- `clearInterval()` itself has no return value: the only result is achieves is making JavaScript stop `setInterval()` from running

```
var interval = setInterval(() => {
    timer()
  }, 1000);

 const timer = () => {
    const dateVar = new Date();
    const time = dateVar.toLocaleTimeString();
    console.log(`timer:${time}`);
  }

  const stopFunction = () =>{
    clearInterval(interval);
  }

  interval; // not a function, a reference
```

While this next example works exactly like the one above, it has a stopMyFunction method. If you ran the function, it would stop the clock: