Repeat an Action at Regular Intervals

Let's get started with web animations!

We'll cover the following

- Kick off a Repeated Action
- Stop a Repeated Action
- Trigger an Action after a Delay

You will be learning how to repeatedly modify an element's content. Here is the associated HTML code and for now the corresponding JavaScript code.

Output

JavaScript

HTML

```
<html>
 <head>
 </head>
 <body>
    <h1 id="title">This page wi
 </body>
</html>
```

It works as expected... Kind of: the countdown never stops. We'll fix this a little later.

Kick off a Repeated Action

How did the previous example work? The JavaScript code defines a function called `decreaseCounter()` that accesses and then decreases one by one the

value of the HTML element named *counter*.

Calling `Number()` in the function code is mandatory: it converts the counter string into a number, which endows it with subtraction functionality.

The call to `setInterval()` triggers a repeated action. This function lets you call a function at regular intervals. Its parameters are the function to call and the time in milliseconds between each call. The returned value is an ID for the repeated action, which can be used to further modify it.

JavaScript

```
// Set up a repeated action
const intervalId = setInterval(callbackFunction, timeBetweenEachCall);
```


Stop a Repeated Action

Let's try to stop the counter once the countdown is complete. We'll also modify the text of the page. Here's the JavaScript code for our example, updated to produce our desired result:

Output

JavaScript

HTML

CSS (SCSS)

```
<html>
 <head>
 </head>
 <body>
        <h1 id="title">This page will self-destruct in <span id="counter">10</span> second(s)
 </body>
</html>
```


In the `decreaseCounter()` function, we only decrease the counter if the current value is higher than 1. If not, we call the function `clearInterval()` and then modify the title of the page. The `clearInterval()` function lets you cut off repeated code execution. It takes as a parameter the ID of the action set by the call to `setInterval()`.

JavaScript

```
// Cancel a repeated action set up with setInterval()
clearInterval(intervalId);
```

Trigger an Action after a Delay

Imagine that you want to modify the page text after its “explosion” in the previous example. You’d modify our example as follows:

Output

JavaScript

HTML

```
// Count down the counter until 0
const decreaseCounter = () => {
  // Convert counter text to a number
  const counter = Number(counterElement.textContent);
  if (counter > 1) {
    // Decrease counter by one
    counterElement.textContent = counter - 1;
  }
  else {
    // Cancel the repeated execution
    clearInterval(intervalId);
    // Modify the page title
    const titleElement = document.getElementById("title");
    titleElement.textContent = "BOOM!!";
    // Modify the title after 2 seconds
    setTimeout(() => {
      titleElement.textContent = "Everything's broken now :(";
    }, 2000);
```

```
  }
};


const counterElement = document.getElementById("counter");

// Call the decreaseCounter function every second (1000 milliseconds)
const intervalId = setInterval(decreaseCounter, 1000);
```


Once the countdown has finished, we call the `setTimeout()` function to set a new page title after a 2 second (2000 millisecond) delay. The `setTimeout()` function lets you execute a function once after a particular delay, expressed in milliseconds.

JavaScript

```
// Execute an action once, after a delay
setTimeout(callbackFunction, timeBeforeCall);
```