A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

Eg:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Function Sequence</h2>

<p>JavaScript functions are executed in the sequence they are called.</p>

<p id="demo"></p>

<script>
function display(some) {
  document.getElementById("demo").innerHTML = some;
}

function myFirst() {
    display("Hello");
}

function mySecond() {
    display("Goodbye");
}

myFirst();
mySecond();
</script>

</body>
</html>
```

Eg 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
</head>
<body>
    <script>

        function display()
        {
            add(100,200);
        }
        function  add(fno,sno)
        {
            document.write("Adding Of Two Numbers :"+(fno+sno));
        }
        display();
    </script>
</body>
</html>
```

Eg3:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Callbacks</h2>

<p>Do a calculation and then display the result.</p>

<p id="demo"></p>

<script>
function myDisplayer(something) {
  document.getElementById("demo").innerHTML = something;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
</script>

</body>
</html>
```

When to Use a Callback?

Where callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

Asynchronous JavaScript

Functions running in parallel with other functions are called asynchronous

Waiting for a Timeout

When using the JavaScript function `setTimeout()`, you can specify a callback function to be executed on time-out:

Eg:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Callback</h2>

<p>Wait 3 seconds (3000 milliseconds) for this page to change.</p>

<h1 id="demo"></h1>

<script>
setTimeout(display, 3000);

function display() {
  document.getElementById("demo").innerHTML = "I Hate You !!";
}
</script>

</body>
</html>
```

Note:

In the example above, `display` is used as a callback.

`display` is passed to `setTimeout()` as an argument.

3000 is the number of milliseconds before time-out, so `display ()` will be called after 3 seconds.

When you pass a function as an argument, remember not to use parenthesis.

Right: setTimeout(`display`, 3000);

Note:

Instead of passing the name of a function as an argument to another function, you can always pass a whole function instead:

Eg:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript SetTimeout()</h2>

<p>Wait 3 seconds (3000 milliseconds) for this page to change.</p>

<h1 id="demo"></h1>

<script>
setTimeout(function() { myFunction("I love You !!!"); }, 3000);

function myFunction(value) {
  document.getElementById("demo").innerHTML = value;
}
</script>

</body>
</html>
```

Waiting for Intervals:

When using the JavaScript function `setInterval()`, you can specify a callback function to be executed for each interval:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript setInterval()</h2>

<p>Using setInterval() to display the time every second (1000 milliseconds).</p>

<h1 id="demo"></h1>

<script>
setInterval(myFunction, 1000);

function myFunction() {
  let d = new Date();
  document.getElementById("demo").innerHTML=
  d.getHours() + ":" +
  d.getMinutes() + ":" +
  d.getSeconds();
}
</script>

</body>
</html>
```