JavaScript assíncrono

‹ Anterior Próximo ›

"Vou terminar mais tarde!"

Funções executadas em paralelo com outras funções são chamadas de assíncronas

Um bom exemplo é JavaScript setTimeout ()

JavaScript assíncrono

Os exemplos usados no capítulo anterior foram muito simplificados.

O objetivo dos exemplos era demonstrar a sintaxe das funções de retorno de chamada:

Exemplo

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

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

myCalculator(5, 5, myDisplayer);
```

Tente você mesmo "

No exemplo acima, `myDisplayer` é o nome de uma função.

Isso é passado `myCalculator()` como um argumento.

In the real world, callbacks are most often used with asynchronous functions.

A typical example is JavaScript `setTimeout()`.

Waiting for a Timeout

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

Example

```
setTimeout(myFunction, 3000);

function myFunction() {
  document.getElementById("demo").innerHTML = "I love You !!";
}
```

Try it Yourself »

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

The function (the function name) is passed to `setTimeout()` as an argument.

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

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

Right: setTimeout(myFunction, 3000);

Wrong: ~~setTimeout(myFunction(), 3000)~~;

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

Example

```
setTimeout(function() { myFunction("I love You !!!"); }, 3000);

function myFunction(value) {
  document.getElementById("demo").innerHTML = value;
}
```

Try it Yourself »

In the example above, `function(){ myFunction("I love You !!!"); }` is used as a callback. It is a complete function. The complete function is passed to setTimeout() as an argument.

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

Waiting for Intervals:

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

Example

```
setInterval(myFunction, 1000);

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

Try it Yourself »

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

The function (the function name) is passed to `setInterval()` as an argument.

1000 is the number of milliseconds between intervals, so `myFunction()` will be called every second.

Waiting for Files

If you create a function to load an external resource (like a script or a file), you cannot use the content before it is fully loaded.

Este é o momento perfeito para usar um retorno de chamada.

Este exemplo carrega um arquivo HTML (`mycar.html`) e exibe o arquivo HTML em uma página da web, depois que o arquivo é totalmente carregado:

Esperando por um arquivo:

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function getFile(myCallback) {
  let req = new XMLHttpRequest();
  req.open('GET', "mycar.html");
  req.onload = function() {
    if (req.status == 200) {
      myCallback(this.responseText);
    } else {
      myCallback("Error: " + req.status);
    }
  }
  req.send();
}

getFile(myDisplayer);
```

Tente você mesmo "

No exemplo acima, `myDisplayer` é usado como retorno de chamada.

A função (o nome da função) é passada `getFile()` como um argumento.

Abaixo está uma cópia de `mycar.html`:

mycar.html

```
<img src="img_car.jpg" alt="Nice car" style="width:100%">

<p>A car is a wheeled, self-powered motor vehicle used for transportation.
Most definitions of the term specify that cars are designed to run primarily on roads, to have
seating for one to eight people, to typically have four wheels.</p>

<p>(Wikipedia)</p>
```

‹ Anterior Próximo ›