Callbacks

Callbacks podem ser resumidas como funções que são passadas como argumentos para outras funções

Elas podem ser chamadas por alguma ação específica, que pode ser um click em um botão HTML, um período de tempo, um retorno de uma API externa, ou simplesmente a execução de uma função.

Temos algumas funções do JavaScript que possuem seus callbacks configurados nativamente, como os métodos de array: filter, find, every, some, reduce.

Na sua criação foram definidos como:
Array.filter(callback)

Uso de Callbacks

No momento de utilizar a função que espera um callback, nós trocamos o parâmetro callback por uma função específica:

Array.filter(callback)

lista.filter((item) => item % 2 == 0)

Uso de Callbacks

Mas também podemos criar nossa função que espera um callback

```
function nome(parametro, callback) {
    // código necessário
    console.log(`Olá ${parametro}`)
    callback()
}
```

Qual é o valor do callback?

Na hora da criação nós não sabemos, mas quem chamar a função vai passar o que é.

Uso de Callbacks

O que for passado como segundo parâmetro vai substituir a linha 4.

```
function nome(parametro, callback) {
    // código necessário
    console.log(`Olá ${parametro}`)
    callback()
}
```

```
nome("mundo", () => console.log("executando o callback") )
```

Na prática a função vira o seguinte:

```
function nome("mundo", callback) {
  // código necessário
  console.log(`Olá mundo`)
  console.log("executando o callback")
}
```

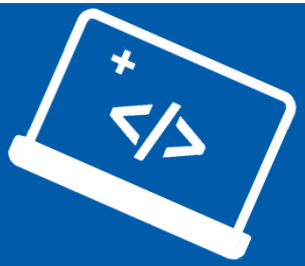
Uso de Callbacks com parâmetros

Podemos passar parâmetros na execução do callback também:

```
function processaAposReceberNome(callback) {
    let nome = prompt("Qual é o seu nome?")
    callback(nome)
}

processaAposReceberNome(nomeQueVaiReceberNaExecucao => {
    alert(`Olá ${nomeQueVaiReceberNaExecucao}`)
})
```

Atividades (callbacks)



- Crie uma função que recebe um callback como argumento:

  Dentro dela crie uma variável nome para receber um nome via prompt e em seguida execute o callback.

  Realize 3 chamadas para essa função passando callbacks diferentes em cada uma, por exemplo:
  1- Passar uma função que exibe um alerta com `Olá ${nome}`
  2- Passar uma função que exibe um alerta com `Feliz aniversário ${nome}`
  3- Passar uma função que exibe um alerta com `Feliz natal ${nome}`

Callback para manipular eventos no HTML.

A função addEventListener permite que sejam associados eventos a elementos HTML e executar algo no JavaScript quando o evento ocorrer.

Exemplo: Criar uma função que será executada quando um item for clicado

```
const botao = document.querySelector("button")

botao.addEventListener("click", () => {
     alert("botão clicado")
})
```

Uma pausa rápida nos callbacks...

JavaScript assíncrono

Um código síncrono tem sua execução na ordem comum das linhas, por outro lado o código assíncrono ocorre quando uma linha é executada foram da ordem comum.

Isso pode ocorrer por uma solicitação a um serviço externo (API, por exemplo) que pode demorar um tempo ou funções que precisam esperar um tempo propositalmente.

Uma das formas de trazer a execução assíncrona no JavaScript é o uso de callbacks em conjunto com a função nativa setTimeout.

setTimeout

O setTimeout permite que uma função espere algum tempo em milissegundos para ser executada.

Isso acaba tirando a execução dela do fluxo comum do JavaScript. Sendo assim, as linhas abaixo dele vão ser executadas, enquanto a função vai ser chamada após o tempo definido.

setTimeout(callback, tempo)

setTimeout - Exemplo

```
console.log('1')

setTimeout(function depoisDeCincoSegundos() {
  console.log('2')
}, 5000)

console.log('3')
```

```
1
3
2
```

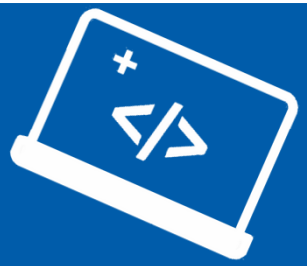
setInterval

A função setInterval repete trechos de código, com um tempo de espera fixo entre cada chamada.

setInterval(callback, tempo)

```
setInterval( () => console.log("rodando código em 2 segundos"), 2000 )
```

Atividades (callbacks)



- Crie uma função dentro de um setTimeout que exibe um alerta com a mensagem “Acesso expirado” após 10 segundos.

- Crie uma função dentro de um setInterval que exibe um prompt perguntando “Tem alguém ai?” a cada 3 segundos.

Outros eventos assíncronos e callback hell

Um evento assíncrono muito importante é a chamada para APIs externas.

Porém o uso de callback nisso torna a sintaxe bastante poluída, podendo formar um inferno de callbacks, que é o nome dado para o aninhamento de diversas callbacks que dependem uma da outra.

Como uma alternativa para esse problema, foi criado um conceito para tornar as operações assíncronas mais legíveis: as Promises

```
// Callback Hell


a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});

```