

https://developer.mozilla.org/pt-BR/docs/Glossary/Callback_function

Callbacks e Promessas

O que são Callbacks?

- Em JavaScript, callbacks são funções passadas como argumento para outra função, que são executadas após a conclusão de uma operação. Isso é útil principalmente em operações assíncronas, como leitura de arquivos, requisições de rede, ou temporizadores.



```
firstTask(data, function(err, result) {  
  secondTask(data, function(err, result) {  
    thirdTask(data, function(err, result) {  
      fourthTask(data, function(err, result) {  
        fifthTask(data, function(err, result) {  
          // Code  
        });  
      });  
    });  
  });  
});
```

```
function saudacao() {  
    console.log("Olá, seja bem-vindo!");  
}  
  
function iniciar() {  
    console.log("Iniciando o sistema...");  
    saudacao();  
    console.log("Sistema iniciado.");  
}  
  
iniciar();
```

A função `iniciar()` chama `saudacao()`, e o código é executado de forma linear. Tudo é executado em sequência sem esperar por ações externas ou demoradas.

```
function saudacao() {  
    console.log("Olá, seja bem-vindo!");  
}  
  
function iniciar(callback) {  
    console.log("Iniciando o sistema...");  
  
    setTimeout(function() {  
        console.log("Concluindo tarefas...");  
        callback(); // Chama o callback após a operação  
    }, 2000); // Simula um atraso de 2 segundos  
}  
  
iniciar(saudacao);
```

A função iniciar() agora recebe uma função callback como argumento.

Usamos setTimeout() para simular uma operação assíncrona que demora 2 segundos.

Após esse tempo, chamamos o callback() para continuar o fluxo, executando saudacao().

```
function operacao1(callback) {  
  setTimeout(function() {  
    console.log("Operação 1 completa");  
    callback();  
  }, 1000);  
}
```


```
function operacao2(callback) {  
  setTimeout(function() {  
    console.log("Operação 2 completa");  
    callback();  
  }, 1000);  
}
```

```
function operacao3(callback) {  
  setTimeout(function() {  
    console.log("Operação 3 completa");  
    callback();  
  }, 1000);  
}  
  
// Chamadas aninhadas  
operacao1(function() {  
  operacao2(function() {  
    operacao3(function() {  
      console.log("Todas as operações concluídas.");  
    });  
  });  
});
```

Ficou difícil de ler ?

- Por isso que até um certo ponto podemos usar o callbacks após isso somente com uso de Promessas (Promises)

javascript

 Copiar código

```
const minhaPromessa = new Promise(function(resolve, reject) {  
  const sucesso = true; // Simulando sucesso ou falha  
  
  if (sucesso) {  
    resolve("A operação foi concluída com sucesso!");  
  } else {  
    reject("Houve um erro na operação.");  
  }  
});  
  
// Consumindo a Promessa  
minhaPromessa  
  .then(function(mensagem) {  
    console.log(mensagem);  
  })  
  .catch(function(erro) {  
    console.error(erro);  
  });
```


O que são Promessas (Promises)?

- **Promises** são objetos em JavaScript que representam o eventual sucesso ou falha de uma operação assíncrona. Elas fornecem uma maneira mais organizada de lidar com código assíncrono, evitando o "callback hell".
- **Estado de uma Promessa:**
- **Pending:** A promessa está aguardando a conclusão da operação.
- **Fulfilled:** A promessa foi resolvida com sucesso.
- **Rejected:** A promessa falhou.

```
javascript Copiar código  
  
const minhaPromessa = new Promise(function(resolve, reject) {  
  const sucesso = true; // Simulando sucesso ou falha  
  
  if (sucesso) {  
    resolve("A operação foi concluída com sucesso!");  
  } else {  
    reject("Houve um erro na operação.");  
  }  
});  
  
// Consumindo a Promessa  
minhaPromessa  
  .then(function(mensagem) {  
    console.log(mensagem);  
  })  
  .catch(function(erro) {  
    console.error(erro);  
  });
```

Time out – Exemplo de Promessa

javascript

 Copiar código

```
function esperarPor(segundos) {  
  return new Promise(function(resolve) {  
    setTimeout(function() {  
      resolve(`Esperou por ${segundos} segundos.`);  
    }, segundos * 1000);  
  });  
}  
  
esperarPor(3).then(function(mensagem) {  
  console.log(mensagem);  
});
```

A função `esperarPor()` retorna uma promessa que é resolvida após um tempo determinado (em segundos).

O código espera 3 segundos e então exibe a mensagem "Esperou por 3 segundos".


```
function operacao1(callback) {
  setTimeout(function() {
    console.log("Operação 1 completa");
    callback();
  }, 1000);
}
```

```
function operacao2(callback) {
  setTimeout(function() {
    console.log("Operação 2 completa");
    callback();
  }, 1000);
}
```

```
function operacao3(callback) {
  setTimeout(function() {
    console.log("Operação 3 completa");
    callback();
  }, 1000);
}
```

```
// Chamadas aninhadas
operacao1(function() {
  operacao2(function() {
    operacao3(function() {
      console.log("Todas as operações concluídas.");
    });
  });
});
```

promises



```
function operacao1() {
  return new Promise(function(resolve) {
    setTimeout(function() {
      console.log("Operação 1 completa");
      resolve();
    }, 1000);
  });
}
```

```
function operacao2() {
  return new Promise(function(resolve) {
    setTimeout(function() {
      console.log("Operação 2 completa");
      resolve();
    }, 1000);
  });
}
```

```
function operacao3() {
  return new Promise(function(resolve) {
    setTimeout(function() {
      console.log("Operação 3 completa");
      resolve();
    }, 1000);
  });
}
```

```
// Encadeamento de Promessas
operacao1()
  .then(operacao2)
  .then(operacao3)
  .then(function() {
    console.log("Todas as operações foram concluídas.");
  });
```

Callbacks

