



Objetivos



1. Explicar o que são Promises
2. Ensinar como manipulá-las
3. Apresentar as palavras-chave “async” e “await”

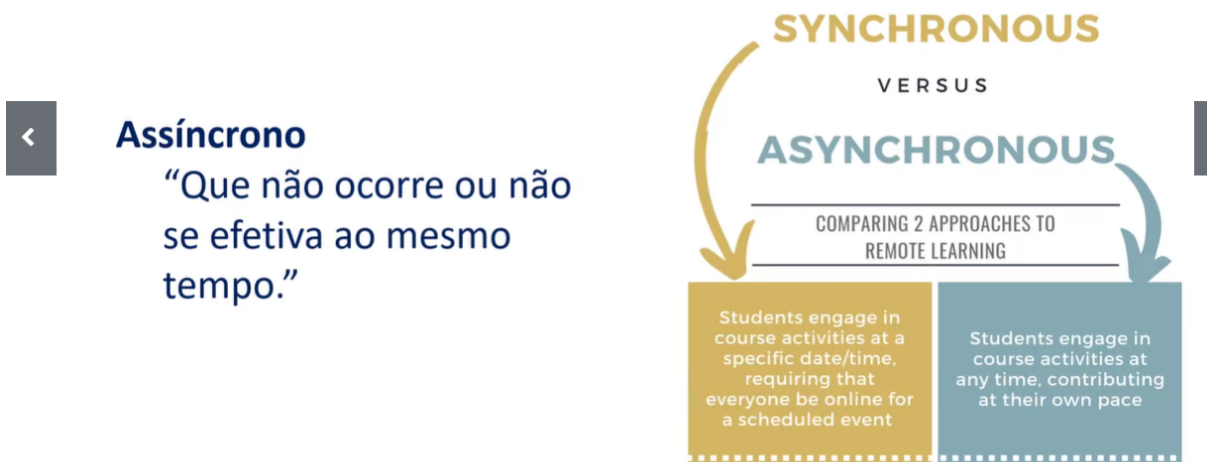
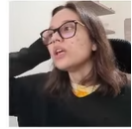


Aula 1 | Etapa 1:

Definição

Javascript Assíncrono

Definição

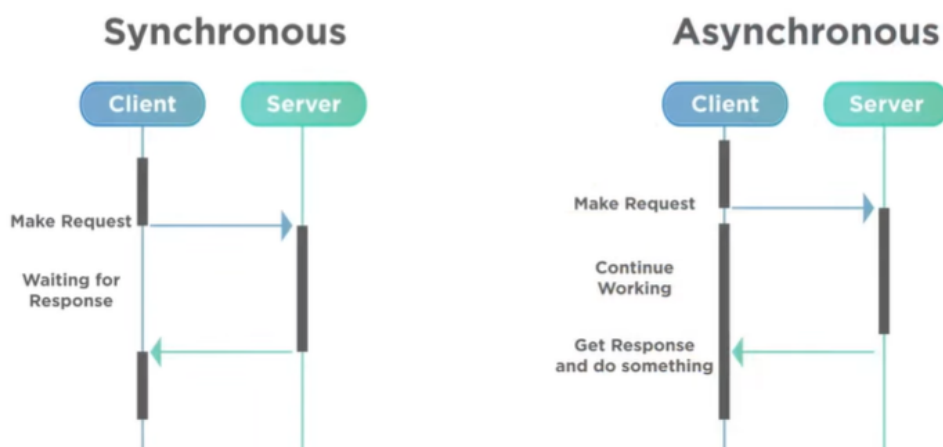


Sincronicidade x Assincronicidade

Um aprendizado síncrono seria: os estudantes vão para o curso numa data específica num tempo específico e todo mundo precisa estar lá naquele momento não fazendo mais nenhuma outra coisa. Assíncrono já é diferente é como a gente está vendo durante a pandemia o aprendizado online, então você vê as aulas do momento que você pode e você pode fazer as coisas no seu próprio ritmo.

Definição

O Javascript roda de maneira **síncrona**.



Então vai acontecer uma coisa depois que essa coisa acontecer a gente começa outro processo e depois que ele termina a gente começa outro. No assíncrono a gente pode fazer uma coisa enquanto faz outra também.



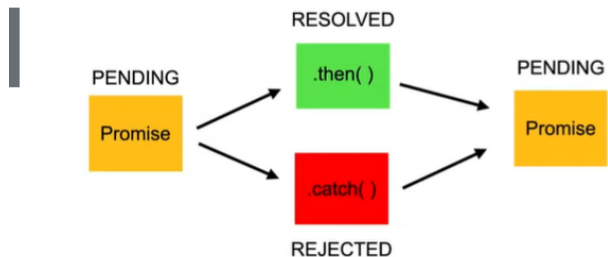
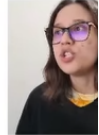
Aula 1 | Etapa 2:

Promises

Javascript Assíncrono

Promises

Objeto de processamento assíncrono



Inicialmente, seu valor é desconhecido. Ela pode, então, ser **resolvida** ou **rejeitada**.

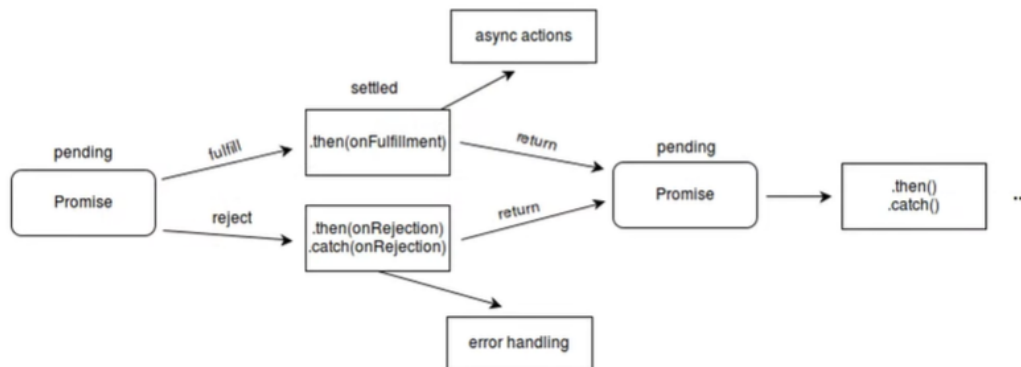
Então uma promise é literalmente uma promessa então vamos fazer uma analogia com um show ou ingresso de cinema, então você compra você vê que o filme vai ser passado naquele dia e você compra o ingresso, o seu ingresso é a promessa de que você vai ver o filme aquele dia e depois de um determinado tempo naquele momento que você comprou o ingresso pode acontecer qualquer coisa depois então você só tem o ingresso você só tem aquilo você não tem o resultado. Então ele pode ser resolvido que é quando acontecer do dia chegar e você ver o filme ou rejeitado algum imprevisto acontece, a sessão é cancelada. Mas no momento que você recebe aquele valor você não sabe o que vai acontecer você não tem o resultado ainda você só vai tê-lo um tempo depois.

É um dado que no momento que você recebe ele você não sabe o valor, mas um tempo depois vai chegar se ele foi resolvido ou não, rejeitado.

Promises

Uma promise pode ter 3 estados

1) Pending 2) Fulfilled 3) Rejected




- 1- Pendente
- 2- Completado
- 3- Rejeitado

Quando você recebe um objeto promise ela sempre está pendente e depois de um tempo ela pode ser rejeitada ou completada.

Promises

Estrutura



```
const myPromise = new Promise((resolve, reject) => {  
  window.setTimeout(() => {  
    resolve(console.log('Resolvida!'));  
  }, 2000);  
});
```

O que fizemos aqui foi que quando eu receber minha promise, depois de 2 segundos eu quero que quando resolver imprima a palavra “Resolvida”. Então eu vou ter a promise eu posso dar um “console.log(myPromise)” e quando ele for executado ele só vai dizer que eu tenho uma promessa nada vai ter acontecido e só depois de 2 segundos eu vou receber o log.

Uma coisa legal que podemos fazer:

Promises

Manipulação

```
const myPromise = new Promise((resolve, reject) => {
  window.setTimeout(() => {
    resolve('Resolvida');
  }, 2000);
});

await myPromise
  .then((result) => result + ' passando pelo then')
  .then((result) => result + ' e agora acabou!')
  .catch((err) => console.log(err.message));

// Após 2 segundos, retornará o valor
// "Resolvida passando pelo then e agora acabou!"
```

A gente pode encadear algumas chamadas então a minha promise primeiro ela vai fazer o que ela tem que fazer e aí o `.then` é o que eu vou fazer depois caso seja bem sucedido a minha promise ou seja caso ela seja resolvida, depois eu vou pegar o resultado dela da promise e eu vou concatenar com uma outra string e eu posso ter mais de uma operação ou seja mais um `then`. E eu vou ter um `.catch` caso dê algum erro e aí eu vou capturar o erro () e mostrar a mensagem do erro.

Então após 2 segundos ele vai retornar “Resolvida” e aí depois eu vou pegar esse valor e vou concatenar com ‘passando pelo then’ e depois eu vou pegar esse ‘Resolvida passando pelo then’ e vou concatenar com ‘e agora acabou’ então ele vai acumular os resultados. Então é isso que acontece, quando uma promise é resolvida você consegue utilizar o resultado dela para fazer outras operações até chegar no resultado final que você espera.

E para você pegar o resultado Depois que a promise é resolvida ou rejeitada você precisa usar a palavra `await`.

Aula 1 | Etapa 3:

Async/await

Javascript Assíncrono

DIGITAL
INNOVATION
LINE

Async/await

Funções assíncronas precisam dessas duas palavras chave.

```
async function resolvePromise() {
  const myPromise = new Promise((resolve, reject) => {
    window.setTimeout(() => {
      resolve('Resolvida');
    }, 3000);
  });

  const resolved = await myPromise
    .then((result) => result + ' passando pelo then')
    .then((result) => result + ' e agora acabou!')
    .catch((err) => console.log(err.message));

  return resolved;
}
```

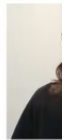
Quando você vai resolver uma Promise você precisa dizer que essa função vai ser assíncrona pra que você possa utilizar a palavra await que é uma palavra criada especificamente para lidar com Promises. Então o await vai parar seu código e ele vai falar assim: Espera essa promise resolver só depois que a gente estiver resolvida o código continuar rodando porque como o Javascript acontece de maneira assíncrona se você não usar o await ele vai falar: “Beleza tem uma Promise aqui não vai resolver e vai continuar o código” então a gente precisa parar o código nesse momento pra dizer: Ó agora tudo vai parar aqui e a gente vai mudar o jeito

que o JS está vendo tudo isso“. Então é para isso que a palavra `await` serve então sempre que você quiser o resultado da sua Promise você precisa colocar o `await`.

Então depois de três segundos eu vou receber a palavra 'Resolvida' abaixo vai acontecer a mesma coisa e eu vou colocar o resultado que vai ser a nossa string numa constante então eu vou armazená-la numa variável constante e depois eu vou retornar o meu valor resolvido.



Async/await



Funções assíncronas também retornam Promises!

```
async function resolvePromise() {
  const myPromise = new Promise((resolve, reject) => {
    window.setTimeout(() => {
      resolve('Resolvida');
    }, 3000);
  });

  const resolved = await myPromise
    .then((result) => result + ' passando pelo then')
    .then((result) => result + ' e agora acabou!')
    .catch((err) => console.log(err.message));

  return resolved;
}
```

```
> resolvePromise()
< ▶ Promise {<pending>}

> await resolvePromise()
< "Resolvida passando pelo then e agora acabou!"
```

Se eu chamar só `resolvePromise` sem o `await`, (isso é uma coisa interessante), uma Promise vai retornar outra Promise. Então você precisa quando você estiver usando uma função assíncrona você não pode simplesmente chamá-la porque a linguagem vai pensar que está pendente você precisa dar um `await` nessa função assíncrona também para receber o resultado, porque o resultado de uma Promise vai ser outra Promise.

Você também pode utilizar o `try/catch` para utilizar as Promises:

Async/await

Utilizando try...catch

```
async function resolvePromise() {
  const myPromise = new Promise((resolve, reject) => {
    window.setTimeout(() => {
      resolve('Resolvida');
    }, 3000);
  });

  let result;

  try {
    result = await myPromise
      .then((result) => result + ' passando pelo then')
      .then((result) => result + ' e agora acabou!')
  } catch(err) {
    result = err.message;
  }

  return result;
}
```

Dentro do try você vai colocar o que eu espero que aconteça caso não dê nenhum erro e dentro do try você não coloca o catch e você vai retornar o resultado mesma coisa.