INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 16 – ASYNC AWAIT

O QUE IREMOS APRENDER

01 FUNÇÃO ASSÍNCRONA

02 PROMISES

03 MÉTODO THEN, CATCH E FINALLY

04 ASYNC AWAIT

05 MÃOS NO CÓDIGO

FUNÇÃO ASSÍNCRONA

Costumamos chamar de programação assíncrona o ato de executar uma tarefa em "segundo plano", sem nosso controle direto disso. Uma função assíncrona, em linguagens de programação como JavaScript, é uma função que é capaz de executar operações assíncronas de forma mais conveniente e legível usando a palavra-chave async. Isso permite que você escreva código que parece síncrono, mas que pode lidar com operações que levam tempo,sem bloquear a execução do programa.

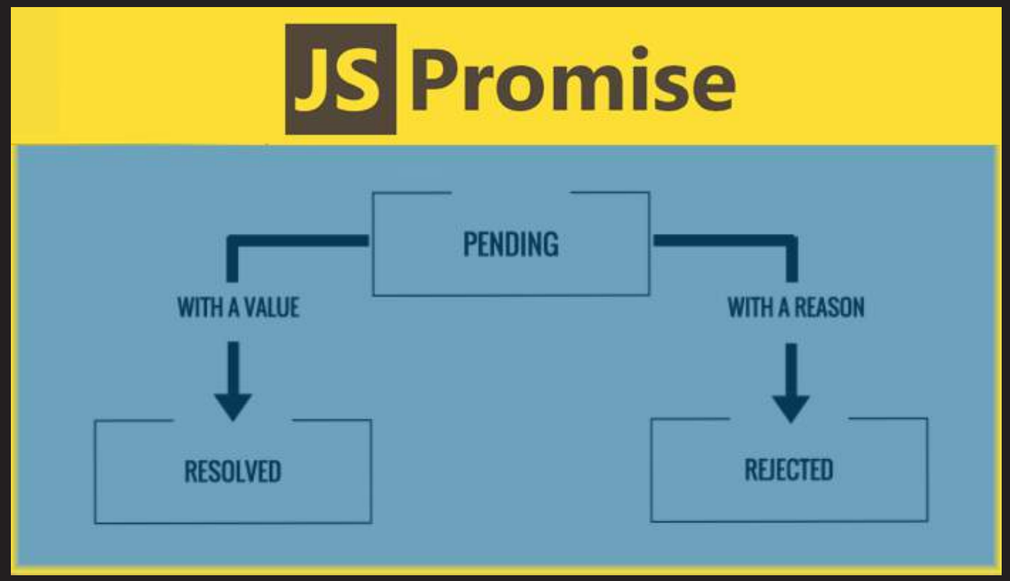
FUNÇÃO SÍNCRONA E ASSÍNCRONA

Função Síncrona: Quando falamos ao telefone, as informações chegam e saem em sequência, uma após a outra; fazemos uma pergunta, recebemos logo em seguida a resposta, com os dados dessa resposta fazemos outro comentário, etc.

Função Assíncrona: uma conversa online via algum mensageiro, como o WhatsApp, enviamos uma mensagem e não ficamos olhando para a tela, esperando, até a outra pessoa responder. Afinal de contas, não temos como saber quando, e se, essa resposta vai chegar.

PROMISES

Promises (promessas) são uma maneira de lidar com tarefas que podem demorar algum tempo para serem concluídas, como fazer solicitações à internet, ler arquivos ou executar operações de banco de dados, em JavaScript. Em vez de bloquear o código enquanto aguardam a conclusão, as Promises permitem que o código continue a ser executado e, quando a tarefa estiver concluída, você pode especificar o que fazer a seguir.



PROMISES

Uma Promise pode estar em um de três estados:

- Pendente (Pending): Isso significa que a tarefa ainda está em andamento e não sabemos se ela terá sucesso ou falhará.
- Realizada (Fulfilled): Isso significa que a tarefa foi concluída com sucesso e retornou um resultado.
- Rejeitada (Rejected): Isso significa que a tarefa falhou e retornou um erro.

PROMISES

Suponha que você queira fazer uma solicitação à internet para obter dados de um servidor. Em vez de bloquear todo o seu código enquanto aguarda a resposta do servidor, você pode criar uma Promise. Quando a resposta do servidor estiver disponível, a Promise pode ser "resolvida" com os dados ou "rejeitada" com um erro, dependendo do resultado.

- Você pode então usar métodos como .then() para especificar o que fazer quando a Promise for resolvida com sucesso e .catch() para lidar com erros se a Promise for rejeitada.

PROMISES

O código ao lado cria uma função chamada addNumbers que recebe dois números como entrada, soma esses números e retorna uma Promise. Se a soma for maior que 0, a Promise é resolvida com o resultado; caso contrário, é rejeitada com uma mensagem de erro.

```
<script>
    function addNumbers(numberA, numberB) {
        return new Promise((resolve, reject) => {
            let result = numberA + numberB;
            if (result > 0) {
                resolve(result);
            } else {
                reject(' O resultado é invalido')
            }

        });
    }
console.log(addNumbers(2,5));
</script>
```

```
▼ Promise {<fulfilled>: 7}
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: 7
>
```

MÉTODO THEN:

O método .then() é usado para lidar com o resultado bem-sucedido de uma Promise.
Quando uma Promise é resolvida (ou seja, sua operação assíncrona é bem-sucedida), o código dentro do .then() é executado.

O método .then() recebe uma função de callback que contém o valor resolvido da Promise, que pode ser usado no código interno.

As chamadas .then() podem ser encadeadas para criar um fluxo de controle assíncrono mais legível e sequencial.

MÉTODO CATCH:

O método .catch() é usado para lidar com erros que podem ocorrer durante a execução de uma Promise.

Quando uma Promise é rejeitada (ou seja, ocorre um erro durante sua operação assíncrona), o código dentro do .catch() é executado.

O método .catch() recebe uma função de callback que contém o motivo da rejeição (geralmente um erro) e permite que você lide com o erro de maneira apropriada.

MÉTODO FINALLY:

O método .finally() é usado para definir um bloco de código que será executado, independentemente de a Promise ser resolvida ou rejeitada.

Isso é útil para ações que você deseja executar após a conclusão da operação assíncrona, independentemente do resultado.

```
function simularSolicitacaoAPI() {
    return new Promise((resolve, reject) => {
        const sucesso = Math.random() < 0.5; // Simula se a opreção foi bem sucedida

        setTimeout(() => {
            if (sucesso) {
                resolve("Dados da API obtidos com sucesso");
            } else {
                reject("Erro ao obter os dados da API")
            }
        }, 1000); // Simula um atraso de 1 segundo na operação
    });
}

// Chamando a função que retorna a Promise

simularSolicitacaoAPI()
    .then(resultado => {
        console.log(resultado);
    })
    .catch(erro => {
        console.error(erro);
    })
    .finally(() => {
        console.log("A solicitação de API foi concluida!")
    });
```

ASYNC AWAIT

Imagine que você está em uma cafeteria e quer fazer um pedido de café. No entanto, a cafeteria é muito popular e, às vezes, a máquina de café leva algum tempo para preparar o seu pedido.

Enquanto espera pelo café, você não quer ficar parado sem fazer nada. Em vez disso, você deseja fazer outras coisas, como verificar seu telefone ou ler um livro. Aqui está como isso se relaciona com async/await:

ASYNC AWAIT

async (Assíncrono): Isso é como você decide que vai fazer um pedido na cafeteria, mas você não quer esperar pessoalmente pelo café. Então, você diz ao atendente da cafeteria que o seu pedido é "assíncrono", o que significa que você não precisa esperar imediatamente pelo café.

await (Aguardar): Agora, você pediu seu café e está livre para fazer outras coisas enquanto a máquina de café prepara o café. Você está "aguardando" que o café fique pronto, mas não precisa ficar parado na cafeteria o tempo todo.

ASYNC AWAIT

Em resumo, async/await é uma forma de lidar com operações assíncronas de forma mais legível e fácil de entender, permitindo que você execute outras tarefas enquanto aguarda o resultado da operação assíncrona.

```
async function fazerPedidoDeCafe() {
    console.log("Fazendo um pedido de café...");
    const cafe = await prepararCafe(); // Aguardando o café ficar pronto
    console.log("Café pronto: ", cafe);
    console.log("Desfrutando do café");
}

function prepararCafe() {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve("CAfé quentinho");
        }, 2000); // tempo de 2 segundos para fazer o café
    });
}

fazerPedidoCafe();
console.log("Fazendo outras coisas enquanto o café é preparado");
```

ATIVIDADE PRÁTICA

Atividade 01

Crie uma função que faz três requisições assíncronas a uma API externa de sua escolha (por exemplo, uma API de notícias, previsão do tempo, etc.). As requisições devem ser feitas em paralelo. Após receber todas as respostas, exiba os resultados no console.

ATIVIDADE PRÁTICA

Atividade 02

Crie uma função chamada obterDetalhesFilme que utiliza a API pública "The Movie Database (TMDb)" para buscar detalhes de um filme específico. Certifique-se de ter uma chave de API válida da TMDb.

A função deve fazer o seguinte:

Faça uma requisição usando fetch para a URL da API para obter os detalhes do filme.

ATIVIDADE PRÁTICA

Atividade 02

Verifique se a resposta da API foi bem-sucedida. Se não, lance uma exceção com uma mensagem de erro.

Exiba os detalhes do filme no console.

Certifique-se de que sua função funcione corretamente e que você tenha substituído 'SUA_CHAVE_DE_API' pela chave de API válida da TMDb. Você também pode escolher um filme diferente ajustando o valor do ID do filme na URL da API.

- Lembre-se de utilizar async/await para lidar com a operação assíncrona de busca de detalhes do filme e tratar erros apropriadamente.

DESAFIO PRÁTICO

Crie uma função chamada obterCotacaoMoeda que usa a API de conversão de moeda da exchangeratesapi.io para obter a taxa de câmbio entre duas moedas. A função deve receber duas moedas como parâmetros: a moeda base (a partir da qual você deseja converter) e a moeda de destino (para a qual você deseja converter).

Fazer uma solicitação à API da exchangeratesapi.io para obter as taxas de câmbio em relação à moeda base

DESAFIO PRÁTICO

Verificar se a resposta da API foi bem-sucedida. Se não, lançar uma exceção com uma mensagem de erro.
Converter a resposta em formato JSON e extrair a taxa de câmbio da moeda de destino.
Verificar se a moeda de destino está presente nas taxas de câmbio. Se não estiver, lançar uma exceção com uma mensagem de erro.
Retornar a taxa de câmbio da moeda de destino.

DESAFIO PRÁTICO

- Em seguida, crie uma função de exemplo chamada exemploConversaoMoeda que use a função obterCotacaoMoeda para converter um valor em uma moeda base para a moeda de destino. Por exemplo, você pode converter um valor em dólares para euros. Exiba o resultado da conversão no console.

- Certifique-se de lidar com erros adequadamente, como a moeda de destino não estar presente nas taxas de câmbio ou uma resposta de erro da API. Use async/await para lidar com operações assíncronas.

DESAFIO PRÁTICO

- Dica: Você pode usar a URL https://api.exchangeratesapi.io/latest?base={moedaBase} para fazer a solicitação à API, onde {moedaBase} é a moeda base especificada como parâmetro.

- Lembre-se de configurar a função exemploConversaoMoeda para usar moedas reais e valores de sua escolha para testar a conversão.

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 16 – ASYNC AWAIT