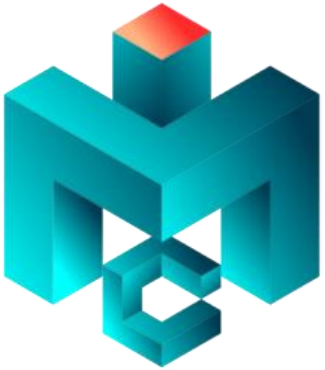


Aula – 9
JavaScript Assíncrono

Disciplina: XDES03 – Programação Web

Prof: Phyllipe Lima Francisco
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação

Agenda

- Conceitos de Programação Assíncrona
- Introdução ao JavaScript Assíncrono
- Promises
- Async/Await

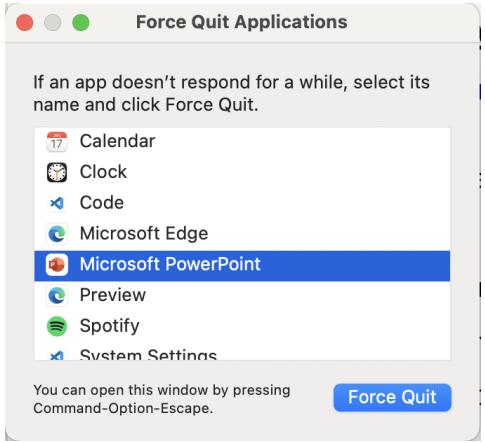
Programação Assíncrona

- Usualmente, as linhas de código de um programa são executadas de forma sequencial.
- Se uma função "X" depende do resultado de outra função "Y", a função "X" precisa esperar o retorno da função "Y". Até que isso se resolva, o programa inteiro fica "parado" da perspectiva do(a) usuário(a).

Instrução 1
Instrução 2
Instrução 3
.....
Instrução N

Programação Assíncrona

- No Mac OSX, por exemplo, podemos ver esse “pause” como o cursor giratório (ou "*beachball*").
- Este cursor é a forma do sistema operacional dizer: “Tem algo demorado ocorrendo, e estou esperando o retorno”



Programação Assíncrona

- Essa situação traz desconforto para o(a) usuário(a), e não faz uso adequado de sistemas com múltiplas CPUs
- O ideia seria deixar essa tarefa “pesada” executando em outra CPU e quando terminar ela “avisa” . Esse situação é a base da programação assíncrona.

Programação Assíncrona Na Web

- Técnicas assíncronas são muito úteis na programação web. Em diversas situações, estamos acessando serviços remotos que podem levar algum tempo para processarem a requisição.
- Nesse contexto, pode ser desejável que outras partes da aplicação web, principalmente a visível para o usuário, permaneça funcionado normalmente.
- Quando a requisição tiver um retorno, ela “avisa” e atualizamos a aplicação com a resposta.

Exemplo 1 – Código Bloqueante

```
const btn = document.querySelector('button');

btn.addEventListener('click', () => {
        let x;
        for(let i = 0; i < 100000000; i++) {
                x = i; //faz nada. Só enrolação
        }

        console.log(x);

        let pElem = document.createElement('p');
        pElem.textContent = 'Eu sou um novo paragrafo!';
        document.body.append(pElem);
});
```

Exemplo 1 – Código Bloqueante

- O código anterior executa uma tarefa demorada que consiste em ficar “enrolando” no laço de repetição.
- Como o código é síncrono, a instrução “console.log” somente será executada após o código concluir o laço de repetição.
- Finalmente, o parágrafo novo somente será criado ao final.

Exemplo 2 – Código Bloqueante e UI

```
const btn = document.querySelector('button');
btn.addEventListener('click', () => {
        let x;
        for(let i = 0; i < 1000000000; i++) {
        x = i*Math.random()*Math.pow(i,2); //faz nada. Só enrolação
        }

        console.log(x);

        let pElem = document.createElement('p');
        pElem.textContent = 'Agora você consegue usar o input!';
        document.body.append(pElem);
});
```

Exemplo 2 – Código Bloqueante e UI

- No Exemplo 2 utilizamos um código semelhante e adicionamos um “input”
- Logo após clicar no botão “clique para travar”, perceba que o “input” não responde imediatamente aos dados que o(a) usuário(a) está inserido.
- Isso ocorrer porque o código está “bloqueado”.

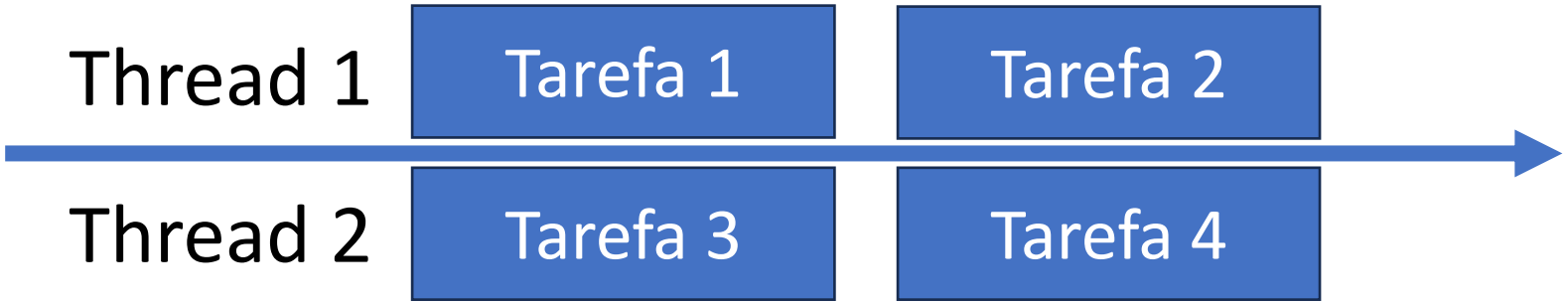
JavaScript e Threads

- Threads são “processos leves” (*lightweight process).*

  Elas representam uma linha de execução e só podem executar uma única tarefa.

JavaScript e Threads

- Um processo pode conter diversas threads, e assim temos o *multithreading*. Se a linguagem de programação suporta *multithreading* podemos usar mais de uma CPU.



JavaScript e Threads

- JavaScript é uma linguagem *single-threaded!* ☹
- A thread única é conhecida como *main thread*
- No Exemplo 1, tínhamos algo como:

JavaScript e *Web-Workers*

- Para introduzir *multithreading* no JS, foi criada a *Web Worker*
- A ideia é delegar para o navegador cuidar de uma "tarefa pesada" sem bloquear a Main Thread.
- Depois que a tarefa é executada, o navegador devolve o controle para o JS.

JavaScript e *Web-Workers*

- Algumas tarefas "pesadas" o navegador utiliza threads separadas para executar.
- Como exemplo a função *fetch()* para requisitar dados de um servidor é executada pelo navegador em uma *thread* separada da *main thread*.

Temporização com JavaScript

- O JS apresenta alguns métodos tradicionais para execução de código assíncrono através de temporização.
  - *setTimeout()// Executa um bloco após um tempo*
  - *setInterval()// Executa um bloco repetidamente*
  - *requestAnimationFrame()*
    - *Versão moderna do setInterval().*

Exemplo 3 -Temporização com JavaScript

```
const btn = document.querySelector('button');
btn.addEventListener('click', () => {

    //Ligar um temporizador
    setTimeout(() => {
        let pElem1 = document.createElement('p’);
        pElem1.innerText = 'Vou aparecer depois!’;
        document.body.append(pElem1);
    }, 5000); //Será executado 5 segundos após ser acionado.

    //Esse código vai executar antes do código dentro
    //do setTimeout
    let pElem2 = document.createElement('p');
    pElem2.innerText = 'Vou aparecer primeiro!';
    document.body.append(pElem2);

});
```

Exemplo 4 –Relógio com JavaScript

```
const btn = document.querySelector('button');
const p = document.querySelector('p’);

//setInterval define que a função mostraRelogio
//Será chamada a cada 1000 milisegundos (1s)
btn.addEventListener('click', () => {
        setInterval(mostraRelogio,1000);
});

const  mostraRelogio = () => {
        let data = new Date();
        let tempo = data.toLocaleTimeString();
        p.innerText = tempo;
}
```

Exercício 1 - Crônometro

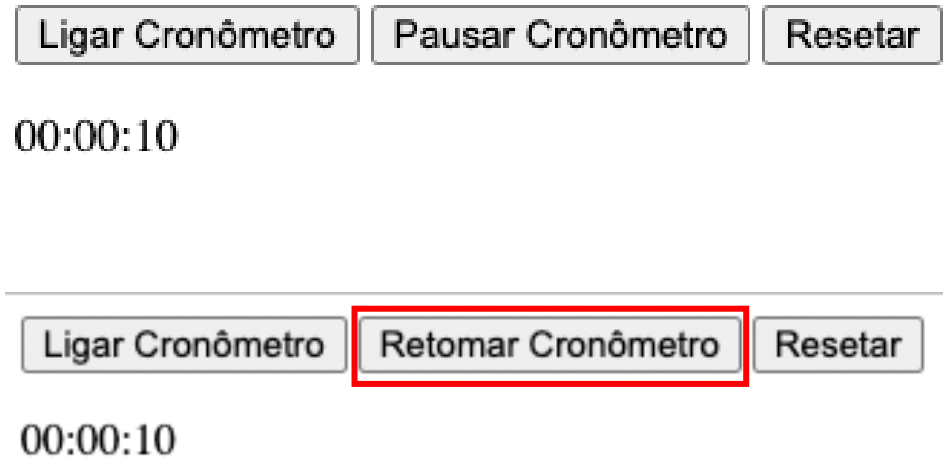
- Construa um cronômetro utilizando a função setInterval().
- Adicione três botões:
  - Iniciar Cronômetro, Pausar, Zerar
- Mostre o tempo decorrido no formato: HH:MM:SS
- Exemplo 1: Se decorrido 123 segundos, o cronômetro deverá apresentar 00:02:03
- Exemplo 2: Se decorrido 450 segundos, o cronômetro deverá apresentar 00:07:30

Dicas Exercício 1

- A função “setInterval” devolve uma referência para essa execução. Esta pode ser armazenada em uma variável e passada como parâmetro para a função “clearInterval”.
- Para formatar um número de acordo com “00:00:00” ou qualquer outro preenchimento consulte a função “padStart”.
- String.prototype.padStart() - JavaScript | MDN (mozilla.org)

Exercício 1 - Crônometro

- Imagem da página web com o cronômetro:



Promise

- *Promises* são a base da programação assíncrona moderna em JavaScript.
- Uma *Promise* é um objeto que representa a eventual conclusão ou falha de uma operação assíncrona.
- Um objeto do tipo Promise pode ter os seguintes estados:
  - Pending => ainda está processando
  - Fulfilled => Significa que a promise foi resolvida
  - Rejected => Significa que a promise teve falha

Promise – then

- Com JS moderno, funções assíncronas retornam objetos do tipo *Promise*.
- Tendo uma referência para esse objeto do tipo *Promise*, podemos "pendurar" uma função que recebe um *callback* em caso de sucesso (promessa será cumprida) e um *callback* em caso de falha (promessa não será mantida).
- A sintaxe é bem simples:

```
//Essa função devolve um objeto do tipo Promise
//Salvaremos uma ref para esse objeto na constante "promise"
const promise = algumaFuncaoAssincrona();

//Caso a promessa seja cumprida "então" faremos alguma coisa
promise.then(callbackSucesso, callbackFalha);
```

Promise – then

- Podemos simplificar ainda mais:

```
//Chamamos diretamente a função que retorna um Promise e "então"
//penduramos os callbacks para caso de sucesso e falha
algumaFuncaoAssincrona().then(callbackSucesso, callbackFalha);

function callbackSucesso(){
console.log("Deu boa!");
}

function callbackFalha(){
console.log("Deu ruim");
}
```

Promise – then e catch

- Podemos usar a sintaxe then-catch e delegar a situação “reject” para o bloco ‘catch’.

```
//Outra forma de encadear uma promise é usando catch, que executará em caso de falha
//Essa notação é mais comum
algumaFuncaoAssincrona()
.then(callbackSucesso)
.catch(callbackFalha);
```

Promise – then encadeado

- Podemos, dentro do "then" retornar outra "promise" e ela, por usa vez ser tratada com um "then" encadeado.

```
algumaFuncaoAssincrona()
        .then((resposta) => {
                return resposta.outraFuncaAssinc();
        })
        .then((retorno) => {
                //faz alguma coisa se outraFunaAssinc tiver sucesso
                //perceba que primeiro algumaFuncaoAssincrona precisa resolver.
                //pois o encadeamento sugere que essa promise depende da primeira
        })
        .catch(erro => {
                //se qualquer promise falhar, o catch será executado
        })
```

Exemplo 5 – Código Fictício

- No material, o código do Exemplo 5 é fictício com os exemplos de promises anteriores para referência de estudo.

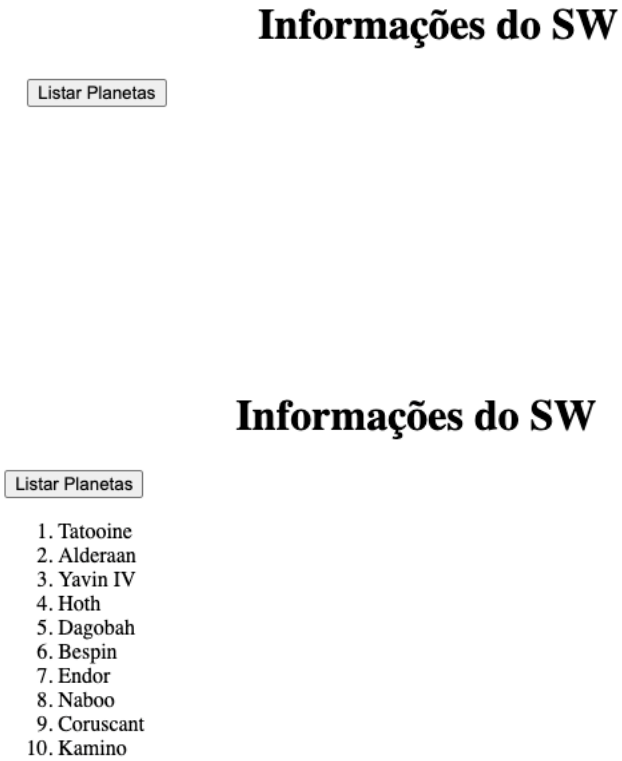
Exemplo 6 – Buscando Recursos do Star Wars

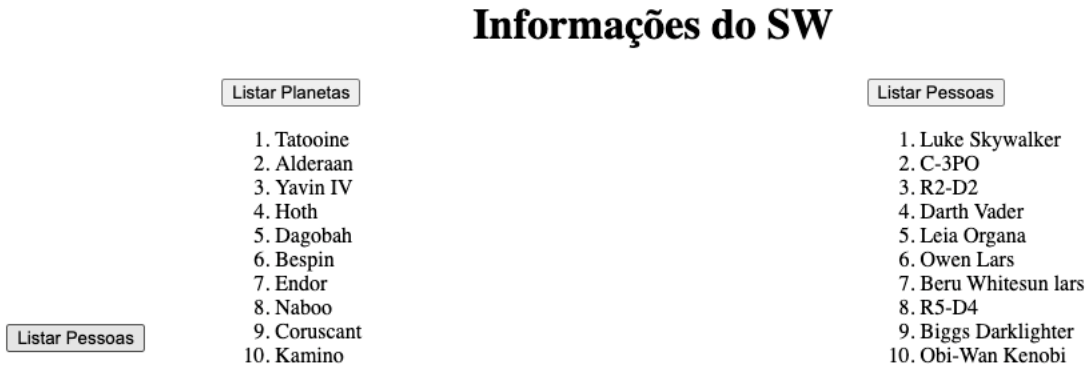
- O método *fetch* da API do JS permite fazermos requisições web.
- Com a resposta, podemos encadear o método json() para extrair os dados em formato JSON. Esse também é uma promise.
- Essa função retorna uma ***promise***, indicando que é assíncrona
- Usando a API do Star Wars: https://swapi.dev/api/ vamos prática promises e requisições.
- A URL: https://swapi.dev/api/planets retorna planetas
- A URL: https://swapi.dev/api/planets retorna pessoas

Exemplo 6 – Buscando Recursos do Star Wars

- Crie uma página web que mostra uma lista com os planetas e as pessoas de star wars.
- Use botões para acionar os métodos fetch()
- Use listas numeradas (<ol>)

Exemplo 6 – Buscando Recursos do Star Wars





Exemplo 6 V2 – Buscando Pokemons

- Usando a API do Pokemon: https://pokeapi.co/ busque os nomes dos pokémon e coloque em uma lista ordenada.
- A URL: https://pokeapi.co/api/v2/pokémon/ retorna o nome dos pokémons.

Exemplo 7 – Buscando Recursos do SW com parâmetros

- Utilizando a URL: https://swapi.dev/api/people/:id/
- Passe um “id” na url para retornar um personagem do SW
- Crie um <input> onde o usuário poderá buscar um personagem pelo seu “id”

Exemplo 7 V2 – Buscando Pokemon com parâmetros

- Utilizando a URL: https://pokeapi.co/api/v2/pokémon/:id/
- Passe um “id” na url para retornar um pokémon.
- Crie um <input> onde o usuário poderá buscar um pokemon pelo seu “id”

Async/Await

- Como uma evolução da legibilidade de código JS, as palavras chaves async/await fornecemum recurso para manipulação de funções assíncronas JS baseada em *promises*.
- Portanto async/await, na prática, forçam funções a retornarem ***promises*** e funcionam normalmente com qualquer função que já retorna ***promise***
- A palavra “async” diz uma função é assíncrona
- A palavra “await” irá esperar o retorno de uma função assíncrona antes de dar sequencia.
- Na prática, o uso de async/await substiui o uso do encadeamento then()

Exemplo 8 – Código Login com Async

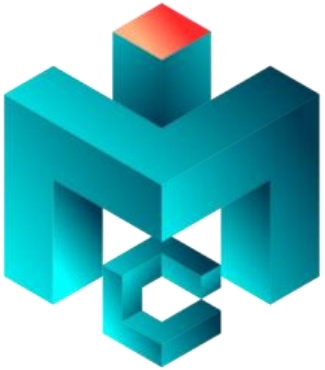
```
const login = async (user,pass) => {
        if(!user || !pass)
                throw 'Credenciais Vazias’;
        if(pass == 'senha’)
                return 'Bem Vindo ao Portal UNIFEI';//poderia ser qualquer valor. Basta retornar um
valor que a promesa será fulfilled
        throw ('Senha Errada’);
        }

login('user', 'senha’)
        .then(msg => {
                console.log('Logado’);
        })
        .catch(err => {
                console.log('Erro’);
                console.log(err);
        })
```

Exemplo 9

- Refaça o Exemplo 7 utilizando async/await

Exercício 2 – Exibir Imagens de Shows de TV

- A URL https://api.tvmaze.com/search/shows?q=busca permite fazer buscas em uma base de dados de séries e programas de televisão passando o query parameter “q”
- Escreva uma aplicação web que apresente as imagens de retorna buscando na URL fornecida.
- Dica: Use o inspector no navegador para encontrar qual a propriedade contém a imagem.
- (show.image.medium)



Aula – 9
JavaScript Assíncrono

Disciplina: XDES03 – Programação Web

Prof: Phyllipe Lima Francisco
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação