PROMISES & FETCH

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA | M1S05-A3

- Revisão
  - Módulos (Export / import)
  - Rest / Spread / Destruct
- Sincronicidade & Assicronicidade
- Promises
- Fetch

REVISÃO | Modules

- Exportando
  - `export const nomeVariavel = 42;`
  - `export default function nomeFuncao() {};`
- Importando (podemos renomear o import default e o comum)
  - `import blah, { nomeVariavel } from './arquivo.js';`
  - `import { nomeVariavel as vName } from './arquivo.js';`
- Incluindo
  - `<script src="app.js" type="module"></script>`

REVISÃO | Rest / Spread / Destruct

- Rest
  - `function nomeFuncao(...paramsArray) {}; //definição`
  - `const {...resto} = objetoExemplo; //no destruct`
- Spread
  - `outraFuncao(...vetor); //invocação`
  - `const objExemplo = { ...outroObjeto };`
- Destruct
  - `const { nome, idade: anos } = objetoPessoa;`
  - `const [ a, b, c, ...r ] = vetorQualquer;`

SINCRONICIDADE & ASSINCRONICIDADE

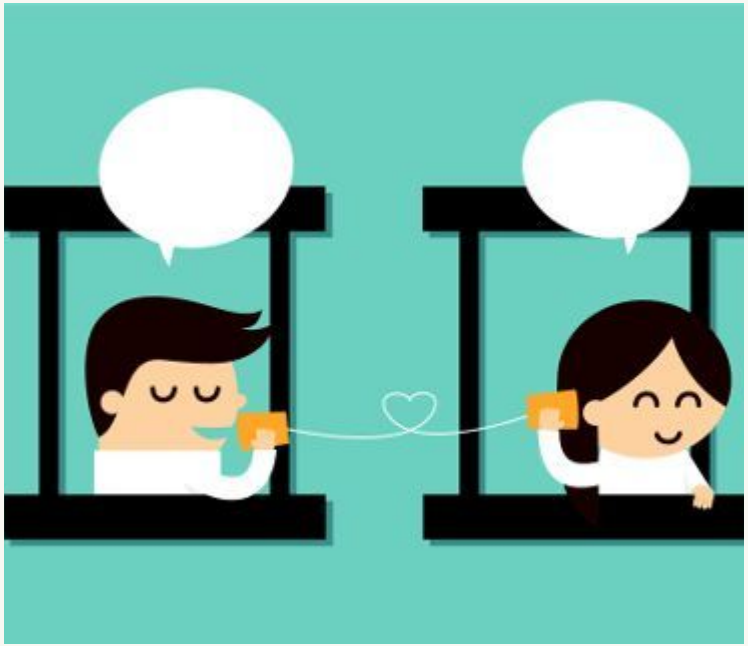
SINCRONICIDADE

- Qualidade ou estado de sincrônico, do que ocorre ao mesmo tempo, em simultâneo.
- Que acontece junto, concomitante.
- Que é síncrono, sem intervalos regulares.

ASSINCRONICIDADE

- Qualidade do que é assincrônico, que não ocorre ou não se efetiva ao mesmo tempo.
- Assíncrono, que não acontece juntamente com outra coisa.
- Que não mantém uma sincronia, com intervalos regulares, por não ser controlado por um sistema temporizador.

SÍNCRONO vs ASSÍNCRONO



EXECUÇÃO ASSÍNCRONA

- Dado este panorama inicial, podemos ter alguns palpites do que seria uma função (ou método) de execução assíncrona?

PROMISES

PROMISES

- O que é uma 'Promise' (ou Promessa) em javascript?

PROMISES

- O JavaScript por si só é tido como uma linguagem que tem que lidar com várias chamadas e execuções que não acontecem no momento em que o script executa em sua inicialização.

PROMISES

- Mas porque utilizar **Promises**?

PROMISES

Um exercício mental:

- Imagine que a Conecta Nuvem lance um novo serviço de empréstimos e você foi encarregado de desenvolver o código desde o pedido do cliente do empréstimo até a entrega do dinheiro em saldo na conta do mesmo.
- Quais são os passos a serem tomados?

PROMISES

- Receber o Pedido de Empréstimo
- Analisar o Perfil do Cliente
- Checar valor disponível
- Verificar se já tem empréstimo ativo
- Liberar o Empréstimo ou não
- Responder se está aprovado ou não

PROMISES

- Agora imaginem que o cliente (navegador) precisa receber a resposta da transação em menos de 3 segundos ou teremos um erro de timeout. Podemos garantir que o cliente receba essa resposta dentro deste limite de tempo?

PROMISES

- Promises definem uma ação que vai ser executada no futuro, ou seja, ela pode ser resolvida (com sucesso) ou rejeitada (com erro).
- Podemos definir funções específicas para lidar com sucessos e falhas.

PROMISES

- Uma promisse sempre estará em algum destes estados:
  - ***pending*** (pendente): 🕛
    Estado inicial, que não foi realizada nem rejeitada
  - ***fulfilled*** (realizada): ✅
    A execução foi concluída com sucesso
  - ***rejected*** (rejeitado): ⛔
    Ocorreu alguma falha na execução

PROMISES | then / catch

```
// exemplo promise
function buscaCEPpromise(cep) {

  return new Promise((resolve, reject) => {
    const resultado = listaCEPs[cep]
    const erro = resultado
      ? null
      : 'CEP inválido!'

    if (erro) {
      reject(erro)
    } else {
      resolve(resultado)
    }
  })
}
```

Exemplo de criação de uma promise

```
// exemplo lidando com resolução e rejeição
buscaCEPpromise('88034001')
  .then(resultado => {
    // caso sucesso
    // exibe resultado
    console.log(resultado)
  })
  .catch(erro => {
    // caso tiver problema
    // exibe erro
    console.error(erro)
  })
```

Definindo funções de sucesso e erro para executar após conclusão

PROMISES | Finally

```
// exemplo promise
function buscaCEPpromise(cep) {

  return new Promise((resolve, reject) => {
    const resultado = listaCEPs[cep]
    const erro = resultado
      ? null
      : 'CEP inválido!'

    if (erro) {
      reject(erro)
    } else {
      resolve(resultado)
    }
  })
}
```

Exemplo de criação de uma promise

```
// exemplo lidando com resoluão e rejeição
buscaCEPpromise('88034001')
  .then(resultado => {
    // caso sucesso
    // exibe resultado
    console.log(resultado)
  })
  .catch(erro => {
    // caso tiver problema
    // exibe erro
    console.error(erro)
  })
  .finally(() => {
    // sempre executa ao final
    console.log('Fim!')
  })
```

Definindo uma função para ser executada ao final de todo processo

PROMISES

```
const promiseUm = buscaCEPpromise('88034001')
const promiseDois = buscaCEPpromise('88034001')

Promise.race([promiseUm, promiseDois])
  .then(resultado => {
    // caso alguma resolva com sucesso
    // exibe resultado da primeira a resolver
    console.log(resultado)
  })
  .catch(erro => {
    // caso nenhuma resolva com sucesso
    // exibe erro
    console.error(erro)
  })
```

Exemplo de concorrência de execução

```
const promiseUm = buscaCEPpromise('88034001')
const promiseDois = buscaCEPpromise('88034001')

Promise.all([promiseUm, promiseDois])
  .then(resultados => {
    // caso todas resolvam com sucesso
    // exibe resultado de todas promises
    resultados.forEach(resultado => {
      console.log(resultado)
    })
  })
  .catch(erro => {
    // caso alguma seja rejeitada
    // exibe erro
    console.error(erro)
  })
```

Exemplo de espera de resultados de execuções paralelas

PROMISES

```
function aguarda3Segundos() {

  return new Promise(resolve => {
    setTimeout(() => {
      // retornará a string para then
      resolve('Aguardou 3 segundos');
    }, 3000); // 3 sec
  });

}
```

Exemplo de promise que aguarde 3 segundos até se resolver

```
aguarda3Segundos()
  .then(retorno => console.log(retorno));

// realizando console.log no momento
// da execução do then
```

Exemplo de espera de resultado da resolução da promise

FETCH

FETCH

```
// o método fetch nos retornará uma promisse
fetch('https://viacep.com.br/ws/88032005/json')
  .then(resposta => {
    // podemos definir uma função then e testar se tivemos sucesso pelo atributo 'ok'
    if (resposta.ok) {
      // o método json da nossa resposta também é uma promise
      resposta.json()
        .then(conteudo => {
          // temos acesso ao nosso conteúdo JSON agora em forma de objeto JavaScript
          console.log(conteudo)
        })
    }
  })
```

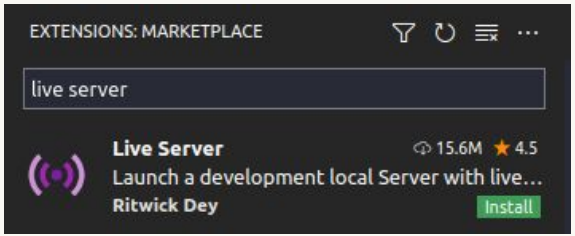
Exemplo de uso de fetch

PARA A MÃO NA MASSA

- Instalar **VS Code**
  (ou outro editor que se sentir mais confortável)
  https://code.visualstudio.com

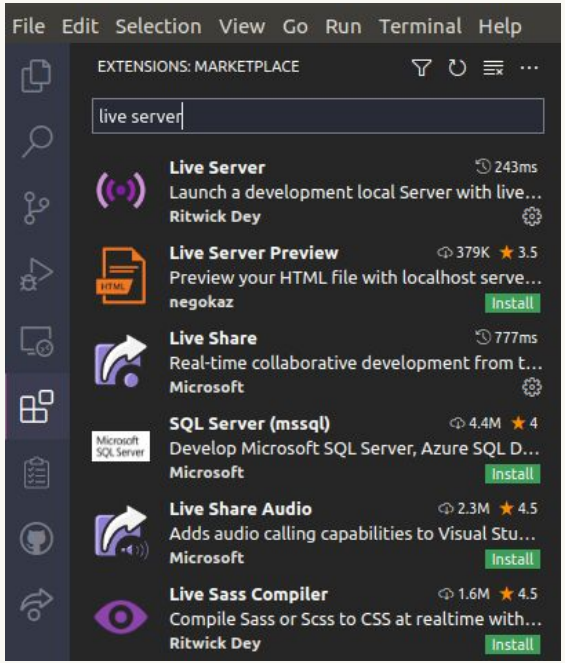
- Sugestão: Instalar extensão **Live Server** no **VS Code**

- Criar um arquivo **index.html** no seu editor

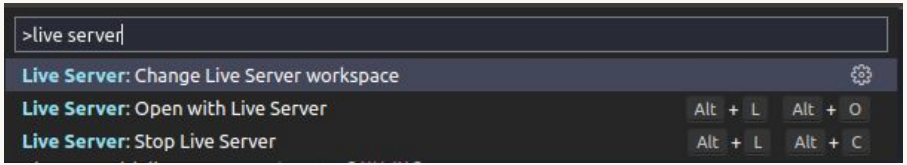


Code Sandbox | https://codesandbox.io
PlayCode | https://playcode.io/new
CodePen | https://codepen.io/pen
JSFiddle | https://jsfiddle.net

PARA A MÃO NA MASSA



- **Ctrl** + **Shift** + **P**
  Live Server: Open with Live Server

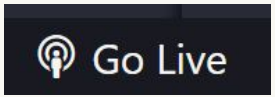


Start

- **Alt** + **L**
- **Alt** + **O**

Stop

- **Alt** + **L**
- **Alt** + **C**



MATERIAL COMPLEMENTAR

Promises em JavaScript: Tudo sobre! | https://youtu.be/Tvbz6u3TyjA

JavaScript Promise in 100 Seconds | https://youtu.be/RvYYCGs45L4

JavaScript Promises In 10 Minutes | https://youtu.be/DHvZLI7Db8E

Asynchronous JavaScript - Callbacks, Promises, Async/Await | https://youtu.be/670f71LTWpM

Learn Fetch API In 6 Minutes | https://youtu.be/cuEtnrL9-H0

06/10 Fetch - JavaScript Antes do Framework (React ou Vue.js) | https://youtu.be/fhIDgAfuJZ8

MATERIAL COMPLEMENTAR

Programação assíncrona | developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Asynchronous/Concepts

JavaScript Assíncrono | https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Asynchronous

Programação com Promises | developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Asynchronous/Promises

Promise | https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/import

Fetch API | https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API

Fetch API e o JavaScript | https://www.braziljs.org/p/fetch-api-e-o-javascript?s=r

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>