Universidade Cruzeiro do Sul

# Programação Web

- JavaScript
- XML
- JSON
- Objeto XMLHttpRequest
- Funções Assíncronas (uso de async e await)
- Fetch API
- Manipulação do XML e JSON no JavaScript e jQuery
- Exemplos
- Exercícios

Professores: Alcides/Cristiane/Ledón

2º Semestre 2023

# O QUE É XML?



XML significa E**X**tensible **M**arkup **L**anguage, linguagem de marcação extensível.

Linguagem muito parecida com o HTML.

É uma linguagem para definir dados, o arquivo XML possui somente os dados, a visualização pode ser feita por diversas outras linguagens.

Geralmente não possui tags definidas, o desenvolvedor cria sua própria definição de tags.

Objetivos de cada linguagem:

- XML foi projetado para transportar e armazenar dados, ou seja, está focado nos dados da aplicação.
- HTML foi projetado para organizar os elementos de uma página Web e estruturar os dados, ou seja, está focado na estruturação da visualização dos dados.

# ONDE USAR XML?



A linguagem XML pode ser usada em diversos tipos de aplicação. É utilizada principalmente em aplicações Web, pois oferece um meio simples de armazenar e compartilhar dados para diversos tipos de linguagens.

É utilizada frequentemente com o AJAX (***Asynchronous JavaScript And XML***).

Se você precisa mostrar os dados dinamicamente no HTML, uma opção seria separar os dados em um arquivo XML e, através do JavaScript, exibir esses dados formatados com HTML/CSS.

Utilizando XML, teremos um arquivo separado com os dados (**arquivo .xml**) e outros para a manipulação e exibição dos dados (**.html, .js e .css**).

Um único arquivo .xml pode ser utilizados em diversas aplicações como, por exemplo:

- JavaScript/HTML/CSS
- Java
- PHP, JSP
- Aplicativos mobile (Android, iOS etc.)
- Etc.

Como utiliza o formato texto, um arquivo XML também pode ser acessado por diferentes tipos de dispositivos, como por exemplo, uma aplicação que roda em smartphone.

# EXEMPLO (LAYOUT DE UMA TELA, APP ANDROID)



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_horizontal" >

    <exemplos.exemplograficos2.MinhaView
        android:id="@+id/minha_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="20dip"
        android:layout_weight="1"/>

    <TextView
        android:id="@+id/digite_text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="Digite um valor de 1 a 5:"
        android:layout_marginBottom="10dip"/>

    <EditText
        android:id="@+id/edita_text"
```

# EXEMPLO BÁSICO



```
<?xml version="1.0" encoding="utf-8"?>
<veiculos>
  <carro>
        <fabricante> FORD </fabricante>
        <modelo> ESCORT </modelo>
        <motorista> Pedro </motorista>
  </carro>
  <carro>
        <fabricante> HONDA </fabricante>
        <modelo> FIT </modelo>
        <motorista> Júlio </motorista>
  </carro>
</veiculos>
```

# OBSERVAÇÕES



Um arquivo XML começa com a linha:

**<?xml version="1.0" encoding="utf-8"?>**

Nesta linha definimos a versão e a codificação de caracteres que será usada para a criação dos dados.

**Um documento XML possui sempre um elemento raiz.** No exemplo anterior, o elemento raiz seria a tag <veiculos>....</veiculos>.

O elemento raiz pode ter vários elementos filhos, como a tag <carro>...</carro> que aparece duas vezes no exemplo.

O elemento carro, por sua vez possui três tags ou elementos filhos, que seriam: <fabricante>, <modelo> e <motorista>.

# ALGUMAS REGRAS NA CRIAÇÃO DO XML



Algumas regras são iguais as utilizadas em XHTML (extensão do HTML com a rigorosidade do XML).

O documento XML deve ter:

- Apenas UM elemento raiz.
- Os atributos devem estar entre aspas.
- Todos os elementos devem ser fechados.
- Os elementos devem estar corretamente aninhados.
- Os nomes das tags são case-sensitive, logo, diferencia letras maiúsculas de letras minúsculas.

# VALIDAÇÃO E APRESENTAÇÃO DO XML



Nosso objetivo é trabalhar com o XML em JavaScript, porém uma pesquisa mais aprofundada em XML seria interessante, pois existem outros tópicos que não iremos abordar aqui, como por exemplo:

- O DTD (*Document Type Definition*) é um documento tipo texto que contém todas as regras estabelecidas para a elaboração de um documento XML qualquer, indicando todas as partes obrigatórias e opcionais que um documento XML deve seguir ao ser criado.

Para manipularmos os dados do arquivo XML podemos utilizar o DOM em JavaScript, o jQuery ou qualquer outro framework que manipule XML.

# ALGUMAS PROPRIEDADES E MÉTODOS DO DOM



Propriedades fundamentais no modelo DOM

- **childNodes** **=> A coleção de nós filhos**
- **firstChild** **=> O primeiro filho**
- lastChild => O último filho
- **nodeValue** **=> O Valor do nó**
- nextSibling => O próximo nó
- previousSibling => O nó anteiror
- nodeName => O nome do nó

Métodos fundamentais do modelo DOM

- **getElementsByTagName(nome)** **ou** **- querySelectorAll(nome)** **=> Captura as tags pelo nome**
- hasChildNodes() => Retorna se existe filhos
- **getAttribute(nome)** **=> Retorna o valor do atributo**

# OBJETO XMLHTTPREQUEST



É um objeto presente no navegador que permite fazer solicitações HTTP em JavaScript.

Ele permite manipular qualquer dado na solicitação HTTP, **não somente XML**.

Permite realizar uma solicitação HTTP a partir de JavaScript ou usando frameworks.

**A leitura do arquivo XML local é bloqueada nos navegadores atuais, como o Chrome. Esses navegadores só permitem a leitura através do protocolo HTTP.**

**Podemos testar usando o Live Server do VS Code**

Temos também o método **fetch()** que é mais moderno e está sendo usado no lugar deste objeto, estaremos estudando nesta aula alguns exemplos.

# COMO CARREGAR UM XML



Crie o objeto

```
let xhr = new XMLHttpRequest();
```

Configurar a conexão

```
xhr.open([get ou post], [URL], [true ou false])
```

O último parâmetro define uma conexão síncrona (false) ou assíncrona (true)

Para definirmos o tipo da resposta do servidor:

```
xhr.responseType = tipo
```

Podemos usar text, document (para XML) e json.

Após configurar, podemos enviar

```
xhr.send()
```

# ALGUNS EVENTOS E PROPRIEDADES



**load** , evento acionado quando a solicitação é concluída
**error** , quando da solicitação não pode ser feita
**progress** , disparado periodicamente enquanto não concluir a solicitação

Após o servidor responder, recebemos as propriedades no objeto:
**status** , código de status HTTP (Ex: 200, 404, etc)
**statusText** , mensagem do status (Ex: OK para 200, Not found para 404)

**A propriedade mais importante:**
**response** , aqui temos a resposta do servidor, pode ser XML, JSON, texto, etc.

# ESTADOS DO OBJETO



O objeto XMLHttpRequest muda de estado a medida que vai processando a solicitação, para sabermos os estados podemos usar a propriedade

**xhr.readyState**

Os estados podem ser:

```
UNSENT = 0;                 // estado inicial
OPENED = 1;                 // solicitação enviada
HEADERS_RECEIVED = 2;       // cabeçalho da resposta recebido
LOADING = 3;                // carregamento da resposta
DONE = 4;                   // requisição finalizada e resposta recebida
```

O objeto percorre os estados na ordem 0 – 1 – 2 – 3 – 4. Sendo que o estado 3 se repete até que a resposta tenha sido recebida.

Podemos analisar a passagem destes estados com a propriedade **xhr.onreadystatechange**

# OBSERVAÇÃO



Os navegadores bloqueiam por segurança o acesso ao objeto XMLHttpRequest localmente, e também bloqueiam o acesso a arquivo de diferentes urls devido a solicitação de recurso de origens diferentes, o famoso CORS (Cross-Origin Resource Sharing).

Obtenha mais informações em https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Controle_Acesso_CORS

**Para rodar nosso exemplo de aula, podemos usar o Live Server do VC Code, simulando assim, um servidor o que nos permite rodar o exemplo sem dificuldades.**

# EXEMPLO

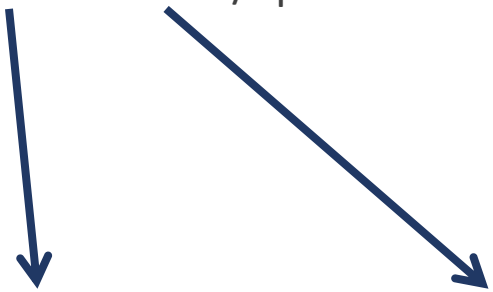


Arquivo XML (menu.xml)

```
<?xml version="1.0" encoding="utf-8"?>

<menu>
    <opcao id="1"> Empresa </opcao>
    <opcao id="2"> Produtos </opcao>
    <opcao id="3"> Contato </opcao>
    <opcao id="4"> Localização </opcao>
    <opcao id="5"> Chat </opcao>

</menu>
```



**atributo id com valor 5**

**valor do nó**

```
<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Document</title>

</head>

<body>

    <div id="menu"></div>
    <script src="js/script15.js"></script>

</body>

</html>
```

Exemplo1.html

# EXEMPLO



```
function criaObjetoXHR(){
    if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
     }
     else {
        alert('A solicitação HTTP não poderá ser efetuada!');

     }
}
```

```
function carregaXML(){
    url = "xml/menu.xml";
    xhr.open('GET', url, true);
    xhr.onreadystatechange = processaRetorno;
    xhr.responseType = 'document';
    xhr.withCredentials = true;
    xhr.send(null);
}
```

```
function processaRetorno(){
    if (xhr.readyState == 4) {
        if (xhr.status == 200) {
            xhrDoc = xhr.response;
            exibirMenu();
        }
    }
}
```

Exemplo1.html

# EXEMPLO



```
function exibirMenu() {
    //carrega os filhos do elemento pai com o nome opcao
    //como existe mais de uma tag com esse nome, é retornado um vetor
    //opc[0], opc[1], opc[2] ...
    opc = xhrDoc.querySelectorAll("opcao");
    alert("Qtde. de elementos retornados = " + opc.length);
    let conteudo="";
    for (let i=0; i< opc.length; i++){
            conteudo += opc[i].getAttribute("id") + " - " + opc[i].childNodes[0].nodeValue + "<br>";
    }
    document.querySelector("#menu").innerHTML = conteudo;
}
```

Início da execução

```
criaObjetoXHR();
carregaXML();
```

Exemplo1.html

# JQUERY E XML



O princípio da manipulação do XML em jQuery é o mesmo do DOM, porém teremos a facilidade que a biblioteca fornece para trabalhar no script.

Lembre-se de adicionar o arquivo .js do jQuery no seu código antes de começar.

Iremos utilizar a função ajax do jQuery

```
$.ajax(url)
 .done(function (xml) {
     //processar retorno
 })
 .fail(function () {
     //mensagem de erro
 });
```

A **url** pode ser uma URL remota real, com domínio, página que gera o XML, JSON etc.

Também, utilizaremos as seguintes funções:

- **find("tag")**
  Obtém todos os itens que tenham esse nome de tag, filtrando por um objeto, seletor jQuery ou elemento, logo teremos um vetor com os elementos retornados.
- **each(<função>)**
  Faz uma repetição sobre o elemento especificado. Em outras palavras seria uma repetição que consegue percorrer os filhos de um elemento.
- **text()**
  Obtém o **conteúdo de texto (valor do nó)** do elemento.
- **attr(<atributo>)**
  Obtém o **valor do atributo** especificado como parâmetro do elemento que está sendo acessado.

# EXEMPLO



Arquivo XML (menu.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu>
    <opcao id="1"> Empresa </opcao>
    <opcao id="2"> Produtos </opcao>
    <opcao id="3"> Contato </opcao>
    <opcao id="4"> Localização </opcao>
    <opcao id="5"> Chat </opcao>
</menu>
```

Obs: colocamos a lógica dentro do carregamento da página, mas poderia ser colocada dentro do evento click de um botão, por exemplo.

**Execute também pelo Live Server.**

```
let url = "xml/menu.xml";
$.ajax(url)
    .done(function (xml) {
        $(xml).find("opcao").each(function () {
            $("#menu").append(
                $(this).attr("id") + "-" +
                $(this).text() + "<br>"
            );
        });
    })
    .fail(function () {
        alert("Ocorreu um erro no carregamento");
    });
```

Exemplo2.html

# JSON (JAVASCRIPT OBJECT NOTATION)



Uma alternativa para o uso de arquivos XML seria o uso do JSON. É um conceito antigo em JS, porém tornou-se popular pela sua facilidade de utilização. A ideia é a mesma do XML, ou seja, poder transportar dados entre aplicações.

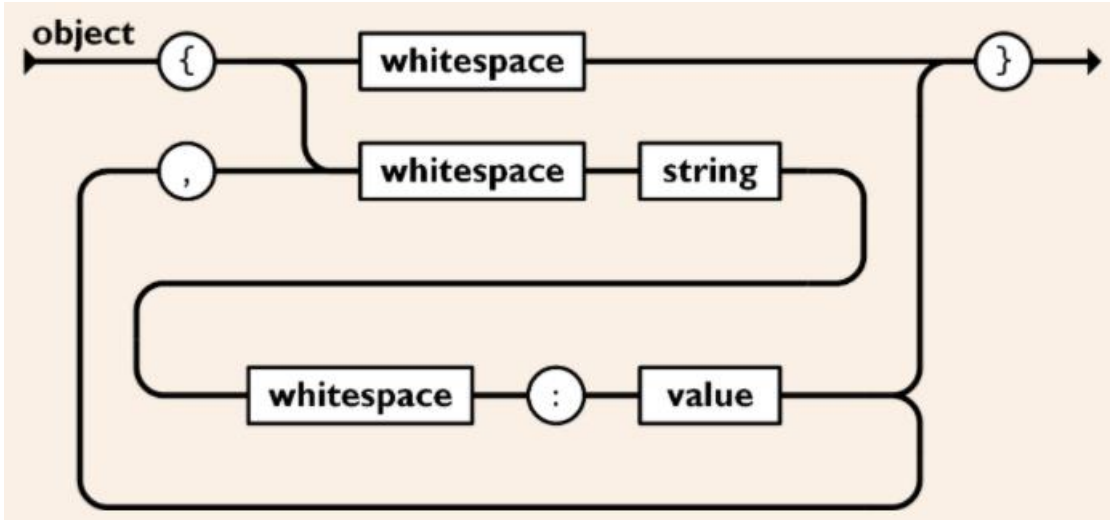
O mesmo grupo de dados em JSON, em geral, utilizará menos memória se comparado com XML e seria mais fácil para interpretar.

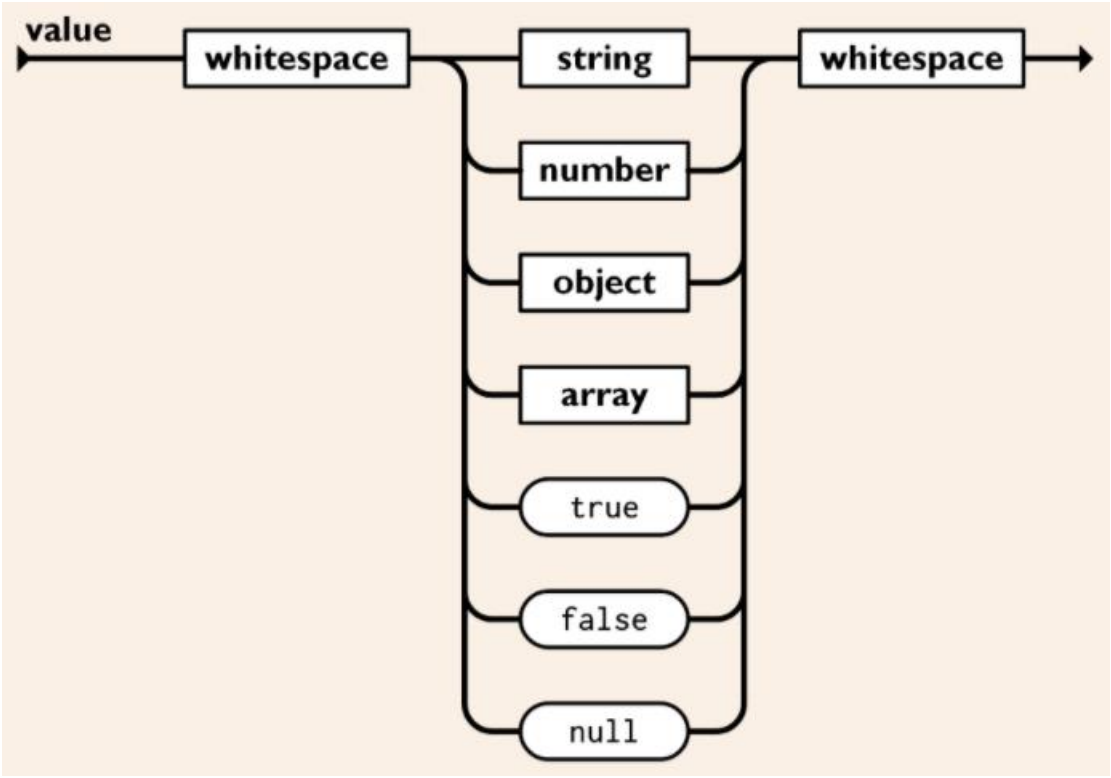
O JSON pode ser utilizado em diversas linguagens, consulte o site http://www.json.org.

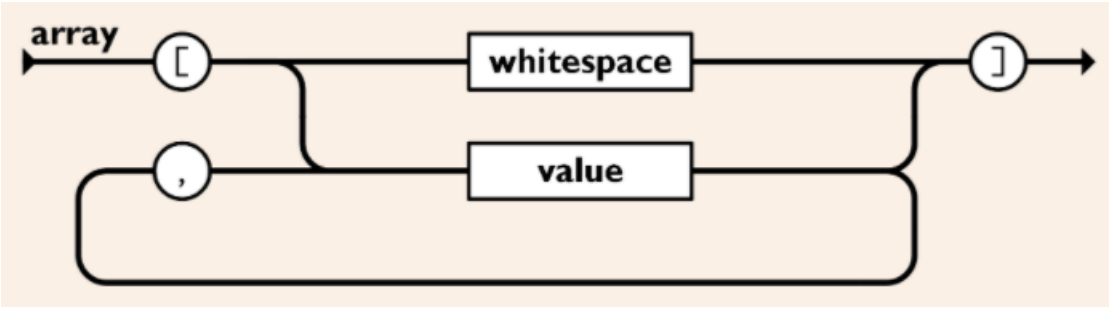
Nosso arquivo JSON pode ter a extensão .json (uso geral em diversas linguagem) ou simplesmente .js (.

# AS ESTRUTURAS EXISTENTES EM JSON






Fonte: https://www.json.org/json-pt.html

# MÉTODOS DO JS



O JavaScript fornece dois métodos básicos para conversão de dados em formato JSON.

- JSON.stringify para converter objetos em JSON.
- JSON.parse para converter JSON de volta em um objeto.

O JSON.stringify(param), converte o parâmetro enviado para uma string chamada de “objeto codificado em JSON”.

Este método aceita diversos tipos como: objetos {...}, vetores [...], strings, números, etc.

O JSON.parse decodifica uma string para o formato JSON.

# XML VS. JSON (EXEMPLO)



Em XML (.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu>
  <opcao id="1"> Empresa </opcao>
  <opcao id="2"> Produtos </opcao>
  <opcao id="3"> Contato </opcao>
  <opcao id="4"> Localização </opcao>
  <opcao id="5"> Chat </opcao>
</menu>
```

Em JSON (.json)

```
{
    "opcoes":
        [
            {"id": 1, "nome": "Empresa"},
            {"id": 2, "nome": "Produtos"},
            {"id": 3, "nome": "Contato"},
            {"id": 4, "nome": "Localização"},
            {"id": 5, "nome": "Chat"}
        ]
}
```

# JSON (EXEMPLO)



Outra alternativa para a criação do arquivo .json seria criá-lo com a extensão .js ou ainda criar um objeto em JavaScript.

```
<script>
let opcoes = [
            {"id": 1, "nome": "Empresa"},
            {"id": 2, "nome": "Produtos"},
            {"id": 3, "nome": "Contato"},
            {"id": 4, "nome": "Localização"},
            {"id": 5, "nome": "Chat"}
        ]
</script>
```

# ARQUIVO JSON COM EXTENSÃO .JSON + JS PURO



(arquivo dados.json)

```
{
   "opcoes":
      [
         {"id": 1, "nome": "Empresa"},
         {"id": 2, "nome": "Produtos"},
         {"id": 3, "nome": "Contato"},
         {"id": 4, "nome": "Localização"},
         {"id": 5, "nome": "Chat"}
      ]
}
```

Obs: a lógica poderia ser colocada dentro do evento click de um botão, por exemplo.

exemplo3.html

```
let xhr;

function criaObjetoXHR(){
    if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
     }
     else {
        alert('Erro');
     }
}
```

script3.js

Rodar com o Live Server

```
function carregaJSON(url){
    criaObjetoXHR();
    xhr.open('GET', url, true);
    xhr.responseType = 'json';
    xhr.withCredentials = true;
    xhr.send(null);
    xhr.onload = function() {
        exibir(xhr.response);
    };
}
```

```
function exibir(json){
    let div = document.querySelector("#menu");
    for(let opc of json.opcoes){
        div.innerHTML += `${opc.id} - ${opc.nome}<br>`;
    }
}

carregaJSON("json/dados.json");
```

# ARQUIVO JSON COM EXTENSÃO .JSON + JQUERY

Rodar com o Live Server

(arquivo dados.json)

```
{
   "opcoes":
      [
         {"id": 1, "nome": "Empresa"},
         {"id": 2, "nome": "Produtos"},
         {"id": 3, "nome": "Contato"},
         {"id": 4, "nome": "Localização"},
         {"id": 5, "nome": "Chat"}
      ]
}
```

```
$.getJSON("json/dados.json", function(data){
    for(let opc of data.opcoes){
        $("#menu").append(opc.id + "-" + opc.nome + "<br>");
    }
});
```

script4.js

Obs: a lógica poderia ser colocada dentro do evento click de um botão, por exemplo.

exemplo4.html

# ARQUIVO JSON COM EXTENSÃO .JS + JS PURO



(uma variável ou em um arquivo externo dados.js)

```
let opcoes = [
    {"id": 1, "nome": "Empresa"},
    {"id": 2, "nome": "Produtos"},
    {"id": 3, "nome": "Contato"},
    {"id": 4, "nome": "Localização"},
    {"id": 5, "nome": "Chat"}
]
```

Se colocamos a estrutura JSON em um arquivo .js externo, devemos usar, por exemplo:

```
<script src="dados.js"></script>
```

```
<body>
    <div id="menu"></div>
    <script src="json/dados.js"></script>
    <script src="js/script5.js"></script>
</body>
```

script5.js

```
let div = document.querySelector("#menu");
for(let opc of opcoes){
    div.innerHTML += `${opc.id} - ${opc.nome}<br>`;
}
```

exemplo5.html

(uma variável ou em um arquivo externo dados.js)

```
let opcoes = [
    {"id": 1, "nome": "Empresa"},
    {"id": 2, "nome": "Produtos"},
    {"id": 3, "nome": "Contato"},
    {"id": 4, "nome": "Localização"},
    {"id": 5, "nome": "Chat"}
]
```

```
<body>
    <div id="menu"></div>
    <script src="js/jquery.min.js"></script>
    <script src="json/dados.js"></script>
    <script src="js/script6.js"></script>
</body>
```

script6.js

```
for(let opc of opcoes){
    $("#menu").append(opc.id + "-" + opc.nome + "<br>");
}
```

exemplo6.html

# FUNÇÕES ASSÍNCRONAS



Funções assíncronas são funções que fazem uma solicitação e aguardam um retorno que pode demorar.

Este tipo de função retorna uma Promise (promessa), este valor pode ser retornado no momento da chamada, pode ser retornado em um momento futuro ou nunca ser retornado. Geralmente o valor da Promise não é conhecido quando a mesma é criada.

Utilizamos a palavra **async** para definir a função.

Sintaxe:

```
async function nome([param, ...]) {
   instruções
}
```

Podemos utilizar na função async, a expressão **await** basicamente pausa a função assíncrona para esperar o retorno da Promise, quando recebe o retorno, o fluxo de execução da função é retomado com o valor da Promise já resolvido.

Em uma requisição de dados via API, temos uma promessa de que estes dados irão chegar, mas enquanto isso não acontece, nosso código precisa continuar rodando. Para isso existem as Promises.

# FUNÇÕES ASSÍNCRONAS



**Ex sem async**:
function soma1(x,y) { return x+y }
console.log( soma1(10,30) )

Retorna: 40

**Ex com async**:
async function soma2(x,y) { return x+y }
console.log( soma2(10,30) )

Retorna: Promise {<fulfilled>: 40}

exemplo7.html

```
function pausa(x) {
    return new Promise(function(resolve) {
      setTimeout(function(){
          return resolve(x)},2000);
    });
}

async function teste1(x){
    let a = await pausa(10);
    let b = await pausa(20);
    console.log(x + a + b);
}

teste1(10);
```

exemplo8.html

# API FETCH



Uma função mais moderna que veio para substituir o objeto XMLHttpRequest.

Permite fazer requisições para buscar dados (XML, JSON, etc) de forma assíncrona pela rede.

Navegadores antigos (+- antes de 2016) não reconhecem esta funcionalidade, nestes casos, devemos usar o objeto XMLHttpResquest.

Uma forma de verificar se o navegador atende é por meio do código

```
if(self.fetch) {
   // solicitação com fetch
} else {
   // solicitação com XMLHttpRequest
}
```

```
Sintaxe básica:

fetch(url)
   .then(function() { })
   .catch(function() { });
```

# CARREGANDO UM XML COM FETCH



```xml
<?xml version="1.0" encoding="utf-8"?>
<universidade>
    <aluno rgm="12345-6">
        <nome>Ana Maria Lopes</nome>
        <idade>31</idade>
        <sexo>F</sexo>
    </aluno>
    <aluno rgm="45454-6">
        <nome>Pedro Souza</nome>
        <idade>21</idade>
        <sexo>M</sexo>
    </aluno>
</universidade>
```

OBS: para ler corretamente o XML, deixe tudo em uma linha única.

```xml
<?xml version="1.0" encoding="utf-8"?>
<universidade><aluno rgm="12345-
6"><nome>Ana Maria
Lopes</nome><idade>31</idade><sexo>F</sexo>
</aluno><aluno rgm="45454-6"><nome>Pedro
Souza</nome><idade>21</idade><sexo>M</sexo>
</aluno></universidade>
```

```js
let url = "xml/alunos.xml";

fetch(url)
    .then((resp) => resp.text())
    .then(str => (new DOMParser().parseFromString(str, "text/xml")))
    .then(function(data) {
        let div = document.querySelector("#lista");
        div.innerHTML = "";
        console.log(data);
        let alunos = data.querySelectorAll("aluno");
        alunos.forEach(function(aluno){
            div.innerHTML += `RGM: ${aluno.getAttribute("rgm")} -
                Nome: ${aluno.childNodes[0].childNodes[0].nodeValue}<br>`
        });
    })
    .catch(function(error) {
        console.log(error);
    });
```

exemplo9.html e exemplo9.js

# CARREGANDO UM JSON COM FETCH



```json
{
   "alunos": [
        {
            "rgm": 12333,
            "nome": "André Lopes",
            "sexo": "M",
            "idade": 34
        },
        {
            "rgm": 12344,
            "nome": "Ana Silva",
            "sexo": "F",
            "idade": 25
        },
        {
            "rgm": 12355,
            "nome": "Luiz Souza",
            "sexo": "M",
            "idade": 51
        }
  ]
}
```

```javascript
let url = "json/alunos.json";

fetch(url)
    .then((resp) => resp.json())
    .then(function(data) {
        let div = document.querySelector("#lista");
        div.innerHTML = "";

        let alunos = (data.alunos);
        console.log(alunos);

        alunos.forEach(function(aluno){
            div.innerHTML += `RGM: ${aluno.rgm},
                              Nome: ${aluno.nome}<br>`;
        });
    })
    .catch(function(error) {
        console.log(error);
    });
```

exemplo10.html e exemplo10.js

# CONSUMINDO UMA API COM FETCH



Exemplo para consumir uma API.
Para o exemplo, utilizaremos a RandomUserAPI (free), https://randomuser.me/
Podemos obter os dados em formado JSON ou XML, utilize:

- XML:
  https://randomuser.me/api/?format=xml&results=10
- JSON:
  https://randomuser.me/api/?format=json&results=10

```xml
<user>
    <results>
        <gender>male</gender>
        <name>
            <title>Monsieur</title>
            <first>Clemens</first>
            <last>Roche</last>
        </name>
        <location>
            <street>
                <number>9582</number>
                <name>Rue ...</name>
            </street>
...
```

```json
[{
        "gender": "female",
        "name": {
            "title": "Ms",
            "first": "Henny",
            "last": "Kösters"
        },
        "location": {
            "street": {
                "number": 9509,
                "name": "Kirchplatz"
            },
...
```

# CONSUMINDO UMA API COM FETCH



```
let url = "https://randomuser.me/api/?format=json&results=10";

fetch(url)
    .then((resp) => resp.json())
    .then(function(data) {
        let div = document.querySelector("#lista");
        div.innerHTML = "";

        let usuarios = data.results;
        console.log(usuarios);

        usuarios.forEach(function(user,i){
            div.innerHTML += `<p><img src='${user.picture.large}'><br>
                              Nome: ${user.name.first}</p>`;
        });
    })
    .catch(function(error) {
        console.log(error);
    });
```

exemplo11.html
exemplo11.js

# JSON + LOCALSTORAGE

Relembrando

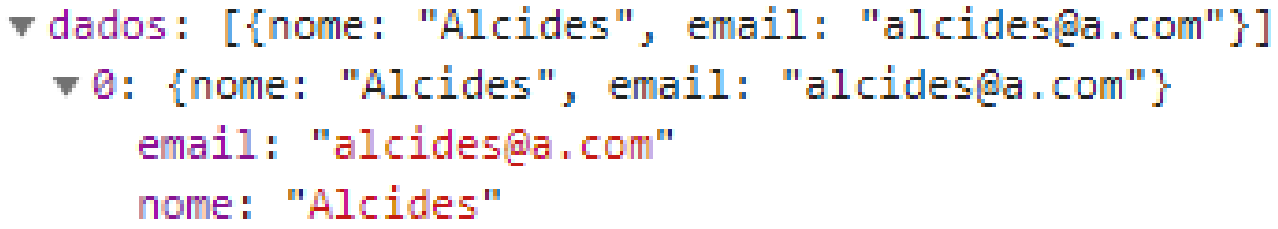
JSON.parse – faz a conversão de uma string para JSON, desde que a string siga o modelo JSON

JSON.stringify – faz a conversão dos valores para uma string JSON.

Ex:

```
base = JSON.parse("{"dados":[{"nome":"Alcides","email":"alcides@a.com"}]}");
```

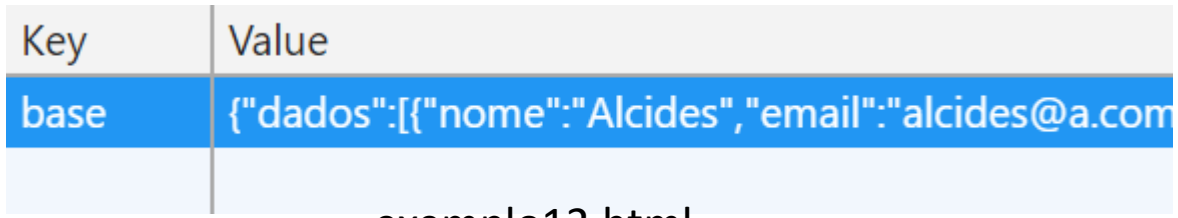
```
JSON.stringify(base);
>>"{"dados":[{"nome":"Alcides","email":"alcides@a.com"}]}"
```



# JSON + LOCALSTORAGE



```html
<body>
    <form>
        Nome: <input type="text" id="nome"><br>
        E-mail:<input type="text" id="email"><br>
        <button type="button" id="btn1">Cadastrar</button>
        <button type="button" id="btn2">Listar</button>
    </form>
    <div id="rel"></div>
    <script src="js/jquery.min.js"></script>
    <script src="js/script5.js"></script>
</body>
```




exemplo12.html

```javascript
let base;
if (localStorage.getItem("base") !== null) {
    base = JSON.parse(localStorage.getItem("base"));
} else {
    base = {
        dados: []
    };
}

$(function () {
    $("#btn1").click(cadastrarDados);
    $("#btn2").click(listarDados);
});

function cadastrarDados() {
    base.dados.push({
                    nome: $("#nome").val(),
                    email: $("#email").val()
                    });
    localStorage.setItem('base', JSON.stringify(base));
}

function listarDados() {
    base = JSON.parse(localStorage.getItem("base"));
    for (let i = 0; i < base.dados.length; i++) {
        $("#rel").append(base.dados[i].nome+"<br>");
    }
}
```