



Front-end: React

Aula 02 - Revisão de JavaScript

Prof. MSc. Kelson Almeida

Agenda

- **Variáveis e tipos de dados**
- **Funções**
- **Arrays**
- **Objetos**
- **Operadores**
- **Condicionais**
- **Loops**
- **Manipulação de DOM**
- **Eventos**
- **API**
- **Conceitos básicos de OO:**
 - Herança
 - Encapsulamento
 - Polimorfismo

Curiosidades sobre JS

- JavaScript foi criado em apenas 10 dias por Brendan Eich em 1995, enquanto ele trabalhava na Netscape Communications Corporation.
- O nome "JavaScript" foi escolhido para capitalizar o sucesso da linguagem de programação Java, que estava em alta na época.
- O JavaScript é uma linguagem de programação interpretada, o que significa que o código fonte é executado diretamente pelo navegador ou aplicativo, sem a necessidade de compilar antes.
- O JavaScript é usado para criar interações dinâmicas e animações em páginas web, além de ser uma das principais linguagens de programação para desenvolvimento de aplicativos web.
- O JavaScript suporta programação orientada a objetos, funcional e procedural, oferecendo aos programadores uma grande flexibilidade no desenvolvimento de seus projetos.

Curiosidades sobre JS

- O JavaScript tem uma ampla variedade de bibliotecas e frameworks disponíveis, como o jQuery, React, Angular, Vue e muitos outros, que ajudam a simplificar o desenvolvimento web e acelerar o tempo de produção.
- O JavaScript é uma das linguagens de programação mais utilizadas em todo o mundo, sendo usada por mais de 95% dos sites ativos na internet.
- O JavaScript é suportado por todos os principais navegadores, incluindo Chrome, Firefox, Safari, Edge e Opera.
- A sintaxe do JavaScript foi influenciada por várias outras linguagens de programação, incluindo Java, C e Perl.
- JavaScript não tem relação com a linguagem de programação Java, apesar do nome similar. As duas linguagens são distintas e têm propósitos diferentes.

Antes de tudo..

- Vamos brincar com o `console.log()` ?



JavaScript debugging

Variáveis

- Variável é nome simbólico para um valor
- Utilizadas para armazenar dados que podem ser usados mais tarde
- Para declarar uma variável em JS podemos utilizar:
 - **var**: Forma mais antiga de se declarar variáveis em JS. Pode ser acessada fora do escopo (caso seja declarada globalmente)
 - **let**: Forma mais moderna de se declarar vars, introduzido no ES6, possuem escopo de bloco, ou seja, só são acessíveis no bloco em que foram declaradas (exemplo: dentro de if/else/funções)
 - **const**: Semelhante ao let em termos de escopo, porém, uma vez declarado valor para ela, não poderá ser reatribuído.



javascript

```
var minhaVariavel;  
let outraVariavel;  
const terceiraVariavel;
```

Tipos de Dados

- Os tipos de dados em JS são divididos em dois tipos principais:
 - Primitivos
 - Objetos
- Fracamente tipada.
- **Tipos de Dados Primitivos:** Valores simples que não tem propriedades métodos.
 - **String:** Sequência de caracteres entre aspas simples ou duplas
 - **Number:** Um número. Inteiros e números de ponto flutuante
 - **Boolean:** Representa um valor lógico
 - **Null:** Valor nulo
 - **Undefined:** Variável que ainda não foi atribuída a um valor

```
javascript

var minhaString = "Olá, mundo!";
let meuNumero = 42;
const meuBoolean = true;
let meuNulo = null;
var minhaIndefinida;
```

Vamos praticar?

- Faça um script que tenha três variáveis:
 - var nome
 - let sobreNome
 - const cpf
- nome e cpf devem estar em um escopo global
- sobreNome deve estar dentro de uma função
- A execução do programa deve imprimir o nome completo na ordem correta (com quebra de linha):
 - **Nome**
 - **Sobrenome**
 - **CPF**



Tipos de Dados

- **Tipos de Dados Objetos:**

Valores complexos que possuem propriedades e métodos.

- **Arrays**

- **Funções**

- **Objetos Regulares**

- **Objetos de Data**

javascript

```
let meuArray = [1, 2, 3];  
var minhaFuncao = function() { console.log("Olá!"); };  
const meuObjeto = { nome: "João", idade: 30 };  
let minhaData = new Date();
```

Funções

- As funções **são blocos de código** que podem ser chamados para executar uma **tarefa específica**.
- Para definir uma função em JS utilizamos a palavra-chave **"function"**
- Seguida pelo **nome da função** e **parâmetros entre parênteses**
- O **corpo** da função é colocado **entre chaves {}** e contém as instruções a serem **executadas** quando a função é chamada.

```
javascript Copy code  
  
function minhaFuncao(parametro1, parametro2) {  
  // corpo da função  
  console.log("O parâmetro 1 é " + parametro1 + " e o parâmetro 2 é " + parametro2);  
}  
  
minhaFuncao("Hello", "World");  
// Output: O parâmetro 1 é Hello e o parâmetro 2 é World
```

Funções

- No exemplo ao lado temos a utilização de um “return” na função. O **return** será responsável por **retornar um resultado final ao processamento da função.**

```
javascript

function somar(num1, num2) {
  return num1 + num2;
}

let resultado = somar(2, 3);
console.log(resultado);
// Output: 5
```

Vamos praticar?

- Faça um script que contenha uma função:
objetivoDaDisciplina(tecnologia)
- A função deve retornar a string:
“Meu objetivo é aprender [parametro tecnologia]”
- Fora da função defina uma **const tecnologia** que possua o valor “React”.
- Chame a execução da função para que a mesma retorne: “Meu objetivo é aprender **React**”



Funções

- Funções também podem ser atribuídas a variáveis e passadas como argumentos para outras funções. São chamadas de funções de ordem superior ou funções de primeira classe em JS.

SCSS

```
let minhaFuncao = function() {  
  console.log("Olá!");  
}  
  
function chamarFuncao(funcao) {  
  funcao();  
}  
  
chamarFuncao(minhaFuncao);  
// Output: Olá!
```

Arrow Function

- Arrow Functions foram introduzidas no ES6.
- São importantes quando funções precisam ser passadas como argumentos para outras funções.
- Observa-se que a arrow func é mais concisa, **não precisamos** utilizar a palavra-chave “**function**”. Usamos o “=>”, entre parênteses colocamos os parâmetros da função, em seguida desenvolvemos o corpo da função.

javascript

```
// Função escrita de forma tradicional
function soma(a, b) {
    return a + b;
}

// Arrow function equivalente
const somaArrow = (a, b) => a + b;
```

Arrow Function

- As Arrow Functions possuem **comportamento diferente** em relação ao **“this”**.
- Em uma **função tradicional** o **“this”** é definido **no momento em que a função é chamada**.
- Na **Arrow Function** o **“this”** é **herdado** do contexto em que a função é definida.
- Observe que em **“falarArrow()”** o **“this”** não é definido dentro do objeto, com isso não foi possível acessar o valor de **“nome”**.
- Em geral, **é recomendado usar arrow functions quando não precisamos usar o “this”** ou quando queremos herdar o valor do **“this”** do escopo externo.

```
javascript Copy code  
  
const objeto = {  
  nome: "Exemplo",  
  falar: function() {  
    console.log("Meu nome é " + this.nome); // Usando o this normalmente  
  },  
  falarArrow: () => {  
    console.log("Meu nome é " + this.nome); // Usando o this da arrow function  
  }  
};  
  
objeto.falar(); // Meu nome é Exemplo  
objeto.falarArrow(); // Meu nome é undefined (pois this não é definido dentro de o
```

Vamos praticar?

- Faça um script que contenha uma função (**ARROW FUNCTION**):
objetivoDaDisciplina(tecnologia)
- A função deve retornar a string:
“Meu objetivo é aprender [parâmetro tecnologia]”
- Fora da função defina uma **const tecnologia** que possua o valor
“React”.
- Chame a execução da função para que a mesma retorne: “Meu objetivo é aprender **React**”



Arrays

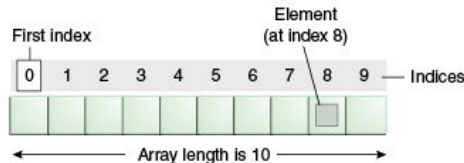
- Em JS, Array é uma **estrutura de dados** que permite armazenar **vários valores em uma única variável**.
- Pode conter qualquer tipo de valor incluindo números, strings, objetos e até mesmo outros arrays.
- A **primeira posição** de um array sempre é indicada pelo **índice “0” (zero)**.



javascript

```
const myArray = [1, 2, 3, 4, 5];
```

Manipulando Arrays



- Podemos criar um array com valores pré-definidos ou criar um array vazio e adicionar os valores posteriormente.
- A propriedade “**push**” vai ser a responsável por adicionar novos elementos a um array.
- Podemos acessar cada elemento individual de um array através do seu respectivo **índice**.

javascript

```
const myArray = [1, 2, 3, 4, 5];
```

SCSS

```
const myArray = [];  
myArray.push(1);  
myArray.push(2);  
myArray.push(3);
```

javascript

```
console.log(myArray[0]); // imprime 1  
console.log(myArray[2]); // imprime 3
```

Manipulando Arrays

- Outras propriedades importantes dos arrays são:
 - **length**: retorna o número de elementos de um array
 - **push()**: adiciona um ou mais elementos no final do array
 - **pop()**: remove o último elemento e retorna-o
 - **shift()**: remove o primeiro elemento do array e retorna-o
 - **unshift()**: adiciona um ou mais elementos no início do array.

JavaScript Array Methods

| | | |
|------------|-----------|-----------|
| pop() | shift() | find() |
| push() | unshift() | forEach() |
| toString() | reverse() | map() |
| join() | concat() | reduce() |
| splice() | slice() | every() |
| sort() | filter() | some() |

Vamos praticar?

- Faça um script que contenha um array chamado notas, esse array armazena 3 notas de um aluno.
- O programa deve imprimir:
 - A primeira nota do aluno é:
...
 - A segunda nota do aluno é: ...
 - A terceira nota do aluno é: ...
 - A média do aluno é: ...

OBS: Cálculo da média: (soma das notas)/quantidade de notas



Objeto

- Em JS, um objeto é uma coleção de propriedades. Cada propriedade é uma chave-valor.
- Ao lado encontramos um exemplo de objeto literal em JS. Um objeto chamado “pessoa”, com três propriedades: nome, idade e cidade. Observe que cada propriedade/chave tem o seu respectivo valor.
- Podemos acessar o valor das propriedades como mostram os exemplos ao lado.
- Também podemos remover por completo uma propriedade presente em algum objeto.

javascript

```
let pessoa = {  
  nome: "João",  
  idade: 30,  
  cidade: "São Paulo"  
};
```

javascript

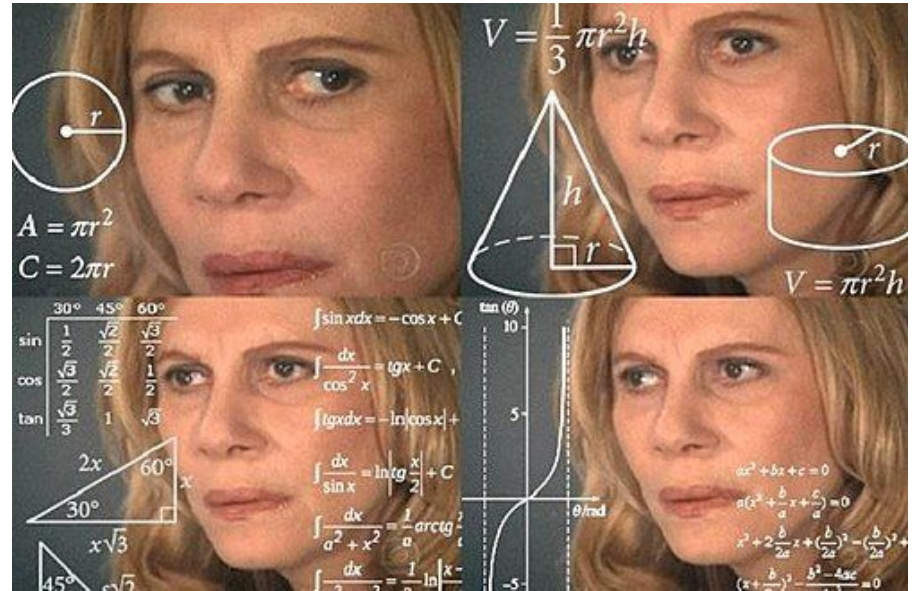
```
console.log(pessoa.nome); // "João"  
console.log(pessoa["idade"]); // 30
```

perl

```
pessoa.email = "joao@gmail.com";  
console.log(pessoa.email); // "joao@gmail.com"  
  
delete pessoa.cidade;  
console.log(pessoa); // {nome: "João", idade: 30, email: "joao@gmail.com"}
```

Operadores

- São símbolos especiais que executam operações matemáticas ou lógicas em valores.
- Alguns tipos de operadores:
 - Operadores aritméticos
 - Operadores de comparação
 - Operadores lógicos
 - Operadores de atribuição



Operadores

- Aritméticos:
 - + (adição)
 - - (subtração)
 - * (multiplicação)
 - / (divisão)
 - % (módulo / resto da divisão)
 - ++ (incremento)
 - -- (decremento)
- Comparação:
 - == (igual a)
 - != (diferente de)
 - > (maior que)
 - < (menor que)
 - >= (maior ou igual a)
 - <= (menor ou igual a)
 - === (igual a em valor e tipo)
 - !== (diferente de em valor ou tipo)

Operadores

- Lógicos:
 - && (e lógico)
 - || (ou lógico)
 - ! (negação lógica)
- Atribuição:
 - = (atribuição simples)
 - += (adição e atribuição)
 - -= (subtração e atribuição)
 - *= (multiplicação e atribuição)
 - /= (divisão e atribuição)
 - %= (módulo e atribuição)

Condicionais

- São utilizadas para executar diferentes blocos de código com base em uma condição.
- Geralmente criadas usando a palavra-chave “if” seguida de uma expressão entre parênteses. Se a expressão for avaliada como verdadeira, o bloco de código dentro das chaves é executado, se não for verdadeira, o bloco é ignorado.

javascript

```
let idade = 18;  
if (idade >= 18) {  
    console.log("Você é maior de idade");  
} else {  
    console.log("Você é menor de idade");  
}
```

Vamos praticar?

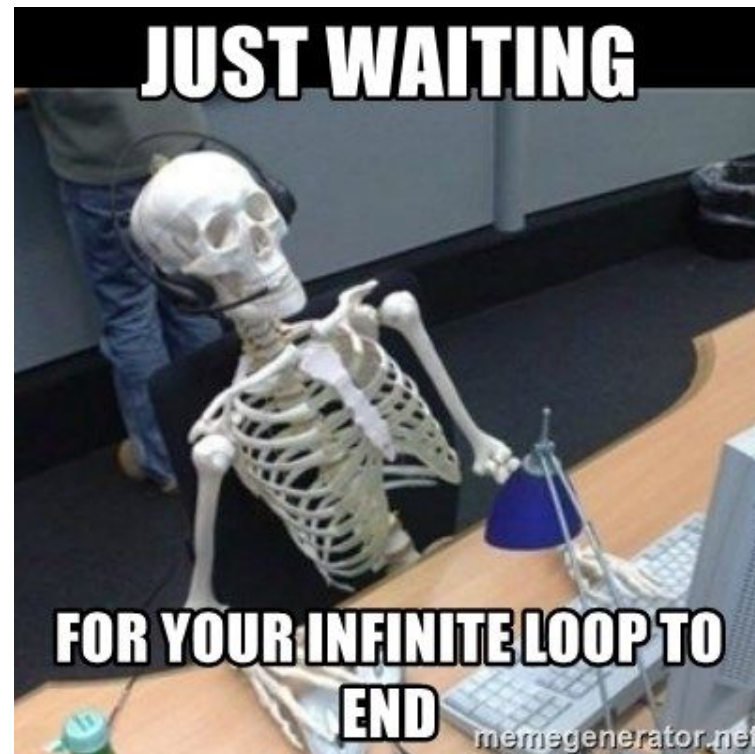
- Faça um script que contenha um array chamado notas, esse array armazena 3 notas de um aluno.
- O programa deve imprimir:
 - A primeira nota do aluno é: ...
 - A segunda nota do aluno é: ...
 - A terceira nota do aluno é: ...
 - A média do aluno é: ...
 - O aluno está: APROVADO ou REPROVADO
 - APROVADO QUANDO A MÉDIA FOR MAIOR OU IGUAL A 7

OBS: Cálculo da média: (soma das notas)/quantidade de notas



Loops

- São uma estrutura de controle em JS que permite repetir a execução de um bloco de código várias vezes.
- Existem 3 tipos de loops em JS:
 - for
 - while
 - do-while



Loop for

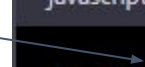
- For é usado quando se sabe quantas vezes deseja repetir o bloco de código.
- Veja a sintaxe básica do for ao lado.

javascript

```
for (inicialização; condição; incremento) {  
    // bloco de código a ser repetido  
}
```

Loop for

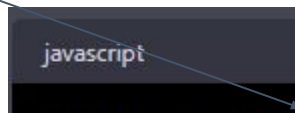
- A **inicialização** é uma expressão que executa apenas uma vez antes do início do loop.
- Utilizada para declarar e inicializar uma variável de controle.



```
javascript
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

Loop for


- A **condição** é uma expressão que é testada no início de cada iteração do loop.
- Se a condição é verdadeira, o bloco de código é executado. Se a condição for falsa, o loop é encerrado.



```
javascript  
  
for (let i = 0; i < 5; i++) {  
  console.log(i);  
}
```

Loop for

- O **incremento** é uma expressão que é executada no final de cada iteração do loop.
- É geralmente utilizada para incrementar a variável de controle.



```
javascript  
  
for (let i = 0; i < 5; i++) {  
  console.log(i);  
}
```

Loop while

- É utilizado quando não se sabe quantas vezes deseja repetir o bloco de código.
- Veja a sintaxe do while ao lado.
- A **condição** é uma expressão que é testada no início de cada iteração do loop.
- Se a condição é verdadeira, o bloco é executado. Se a condição é falsa, o loop é encerrado.

```
javascript

let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```


Vamos praticar?

- Escreva um programa que imprima os números de 1 a 100. Mas, para múltiplos de 3, imprima "Fizz" em vez do número e, para múltiplos de 5, imprima "Buzz". Para números que são múltiplos de ambos 3 e 5, imprima "FizzBuzz".
- Dica: Exemplo para verificar se um número é múltiplo de 3:
 - **número % 3 == 0**



Manipulação de DOM

- Manipulação de **DOM** (Document Object Model) é uma técnica usada em programação web para modificar o conteúdo, a estrutura ou estilo de uma página web após ela ser carregada pelo navegador.



Manipulação de DOM

- Mas o que é DOM, professor?
- DOM é uma representação, em memória, da estrutura da página web.
- Estrutura essa que é criada pelo navegador a partir do código HTML enviado pelo servidor web.



Manipulação de DOM

- A manipulação de DOM é realizada principalmente utilizando JavaScript. Existem várias maneiras de manipular a DOM com JavaScript, incluindo:
 - **Selecionar elementos:**
 - `document.getElementById()`
 - `document.getElementsByClassName`
 - `document.getElementsByTagName()`
 - `jQuery()`



Manipulação de DOM

- A manipulação de DOM é realizada principalmente utilizando JavaScript. Existem várias maneiras de manipular a DOM com JavaScript, incluindo:
 - **Adicionar ou remover elementos:**
 - .createElement()
 - .appendChild()
 - .removeChild()
 - .replaceChild()



Manipulação de DOM

- A manipulação de DOM é realizada principalmente utilizando JavaScript. Existem várias maneiras de manipular a DOM com JavaScript, incluindo:
 - **Modificar conteúdo:**
 - .innerHTML
 - .textContent



Manipulação de DOM

- A manipulação de DOM é realizada principalmente utilizando JavaScript. Existem várias maneiras de manipular a DOM com JavaScript, incluindo:
 - **Modificar estilos:**
 - backgroundColor
 - color
 - fontSize



Manipulação de DOM

- A manipulação de DOM é uma técnica poderosa e flexível, que permite aos devs criar páginas web dinâmicas e interativas. Porém, se utilizada em excesso pode levar a um desempenho lento da página.
- Por isso, é importante usá-lo com cuidado e de forma eficiente! :)



Vamos praticar?

- Suponha que temos a seguinte página HTML
- Crie um script em JS (no arquivo script.js) que tenha uma função “mudaTexto()”, ela deve ser responsável por mudar o texto do elemento <h1> para “Novo Título” quando o botão for clicado.
- Dica: para selecionar o elemento você pode utilizar document.getElementById(“id-do-elemento”)
- Com esse elemento selecionado você pode mudar o texto: variavelQueSelecionouOElemento = innerText = “Novo Título”



```
<!DOCTYPE html>
<html>
  <head>
    <title>Manipulação de DOM</title>
  </head>
  <body>
    <h1 id="titulo">Título da Página</h1>
    <p id="paragrafo">Este é um parágrafo de exemplo.</p>
    <button id="botao" onclick="mudaTexto()">Clique Aqui</button>
    <script src="script.js"></script>
  </body>
</html>
```

Eventos em JS

- **Eventos** em JavaScript são ações ou ocorrências que acontecem dentro de uma página web, por exemplo, como o **clique em um botão**, a **digitação em um campo** de uma formulário, **carga da página** ou a **mudança de estado** de um elemento.

javascript

```
const meuBotao = document.getElementById('meu-botao');

meuBotao.addEventListener('click', function() {
  // código para executar quando o botão for clicado
});
```

Eventos em JS

- O JavaScript permite que você **capture esses eventos** e crie **respostas personalizadas** a eles.
- Por exemplo, você pode criar uma função que será executada quando um usuário clicar em um botão.
- Existem muitos tipos de eventos em JS, incluindo **eventos do mouse (cliques e movimentos)**, **eventos do teclado (ex: pressionamento de teclas)**, **eventos de formulários (envio e reset)**, **eventos de página (carregamento / descarregamento)**, **eventos de animação (início e término de animações)** e muitos outros.

javascript

```
const meuBotao = document.getElementById('meu-botao');  
  
meuBotao.addEventListener('click', function() {  
    // código para executar quando o botão for clicado  
});
```

Eventos em JS

- Para capturar um evento em JS, você precisa adicionar um **ouvinte** de eventos ao elemento HTML correspondente.
- O método **addEventListener()**, no exemplo ao lado, vai capturar o evento de clique em um botão.
- Neste exemplo, o “**getElementById()**” é utilizado para selecionar o elemento HTML com o ID “**meu-botao**”, em seguida um ouvinte de eventos é adicionado a ele usando o método “**addEventListener()**”.
- A função passada como segundo argumento será executada sempre que o botão for clicado.

javascript

```
const meuBotao = document.getElementById('meu-botao');  
  
meuBotao.addEventListener('click', function() {  
    // código para executar quando o botão for clicado  
});
```

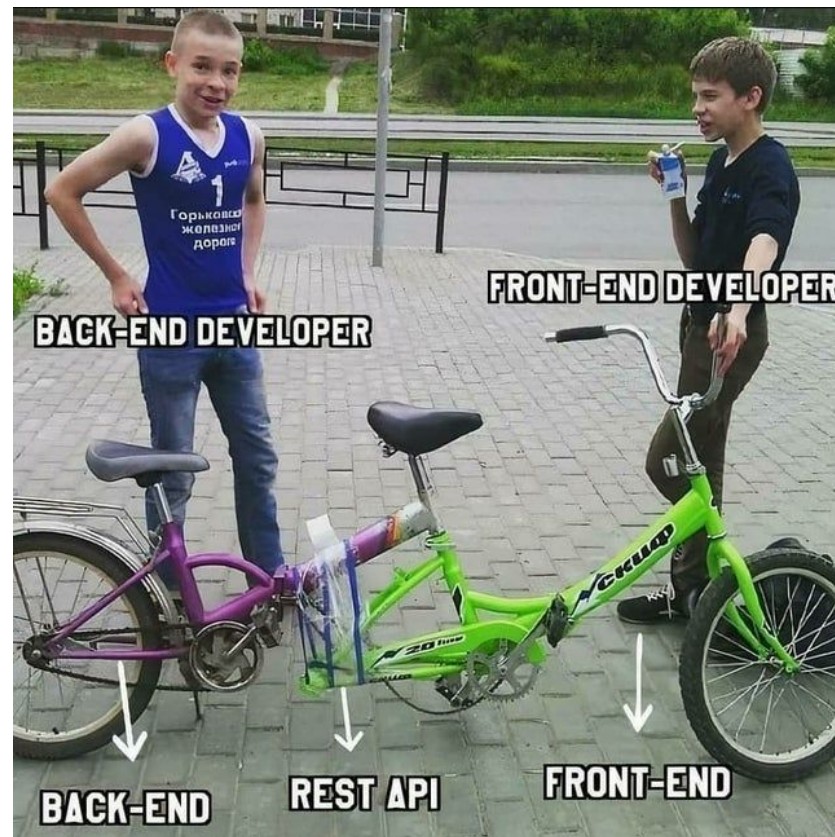
Vamos praticar?

- Escreva um html que contenha um botão “Clique aqui”
- Dentro desse HTML inicialize um “<script> ... </script>” que contenha um script em JS que receba esse elemento do botão em uma variável, e logo após implemente um “ouvinte” com essa variável para exibir o alert “Botão clicado!”
 - `alert(“Botão clicado!”);`



O que é API, prof???

- **Application Programming Interface.**
 - Interface de Programação de Aplicação
- **Conjunto de normas que possibilita a comunicação entre plataformas**
 - Através de protocolos e padrões
- Situação: *“Meu back precisa se comunicar com o meu front!”* - API nele!



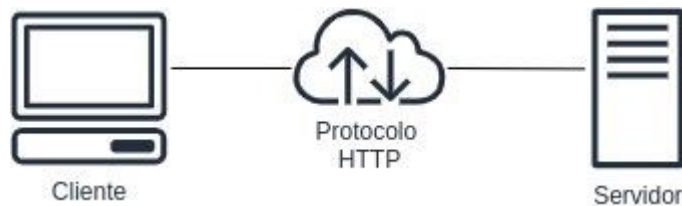
Rest API? RESTful?

- Também chamada de API **RESTful**.
- Interface de programação de aplicações.
 - Em conformidade com a arquitetura **REST**.
- **REST**: Representational State Transfer. É um **conjunto de princípios** que devem ser seguidos ao projetar um sistema web.
- **RESTful**: é a **forma de implementar** os princípios ditados na arquitetura REST.



O que é HTTP?

- **Hypertext Transfer Protocol**
 - Camada de Aplicação do modelo OSI.
- **REGRAS** da comunicação entre o cliente (ex: Navegador) e um servidor na internet.
- **Requisição (request):** Todo pedido que é enviado ao servidor.
- **Resposta (response):** Resposta do servidor, seguindo o fluxo do request.



Verbos HTTP

- Utilizados no desenvolvimento e/ou no consumo de serviços RESTful.
- Objetivo: o serviço vai prover uma URL base e os verbos HTTP tem a responsabilidade de indicar a ação que é requisitada pelo consumidor do serviço em questão.

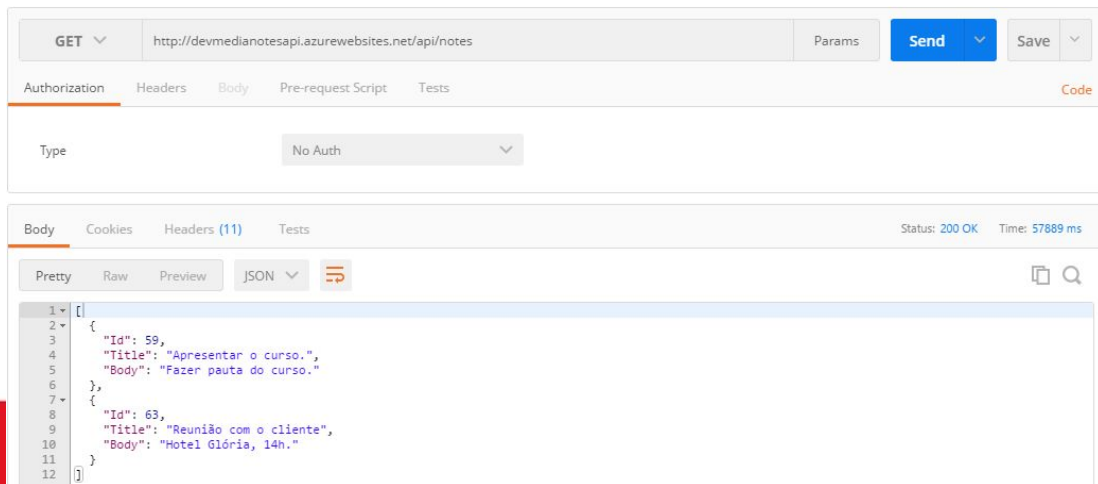
| | | |
|--------|--------------------------|------------------------|
| GET | /pet/{petId} | Find pet by ID |
| PUT | /pet | Update an existing pet |
| DELETE | /pet/{petId} | Deletes a pet |
| POST | /pet/{petId}/uploadImage | uploads an image |

/books

| | | |
|--------|-----------------|-------------------------------------|
| GET | /books | Lists all the books in the database |
| DELETE | /books/{bookId} | Deletes a book based on their id |
| POST | /books | Creates a Book |
| PUT | /books/{bookId} | Method to update a book |
| GET | /books/{bookId} | Retrieves a book based on their id |

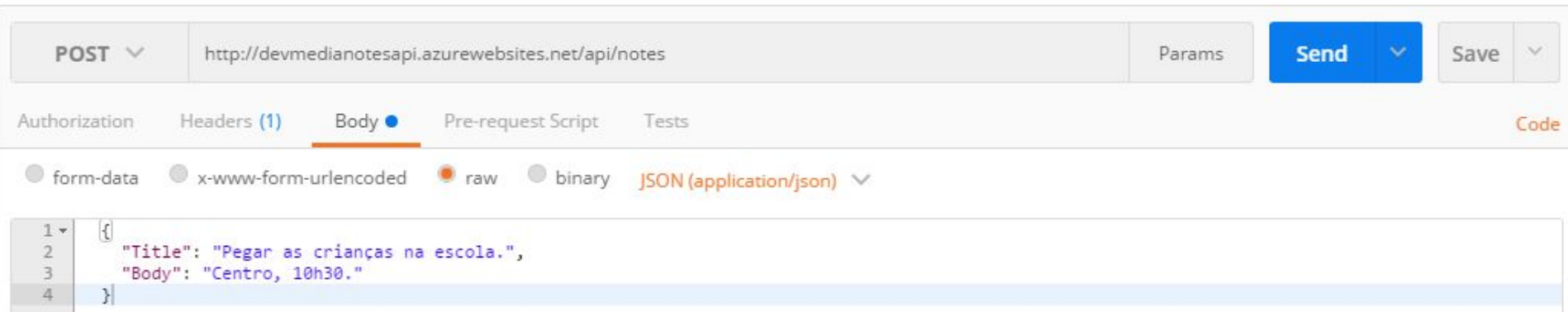
Verbos HTTP

- **GET**
- Solicita a representação de um recurso específico.
- Requisições com GET retornam apenas dados.
- Ler dados, jamais alterar!



Verbos HTTP

- **POST**
- Criação de recursos
- Adiciona informações a um recurso
- Geralmente é utilizado para enviar dados de formulários



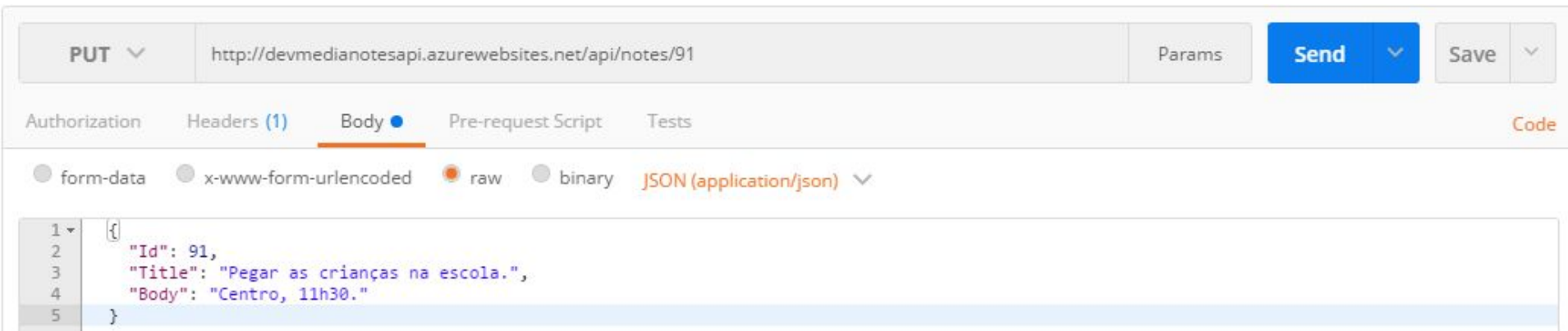
The screenshot shows a REST client interface with the following details:

- Method:** POST (selected from a dropdown)
- URL:** `http://devmedianotesapi.azurewebsites.net/api/notes`
- Params:** A button labeled "Params" with a dropdown arrow.
- Buttons:** A blue "Send" button with a dropdown arrow, and a "Save" button with a dropdown arrow.
- Tabs:** Authorization, Headers (1), Body (selected), Pre-request Script, Tests. A "Code" link is visible on the right.
- Body Type:** A row of radio buttons for `form-data`, `x-www-form-urlencoded`, `raw` (selected), and `binary`. A dropdown menu shows `JSON (application/json)`.
- Body Content:**

```
1 {  
2   "Title": "Pegar as crianças na escola.",  
3   "Body": "Centro, 10h30."  
4 }
```

Verbos HTTP

- **PUT**
- Atualiza informações de um recurso (update)



The screenshot shows a REST client interface with the following elements:

- Method:** PUT (selected from a dropdown)
- URL:** http://devmedianotesapi.azurewebsites.net/api/notes/91
- Params:** A button to manage query parameters.
- Send:** A blue button to execute the request.
- Save:** A button to save the request.
- Tabs:** Authorization, Headers (1), Body (selected), Pre-request Script, Tests.
- Body Type:** radio buttons for form-data, x-www-form-urlencoded, raw (selected), and binary.
- Content Type:** JSON (application/json) (selected from a dropdown).
- Body Content:** A text area containing a JSON object:

```
{
  "Id": 91,
  "Title": "Pegar as crianças na escola.",
  "Body": "Centro, 11h30."
}
```

Verbos HTTP

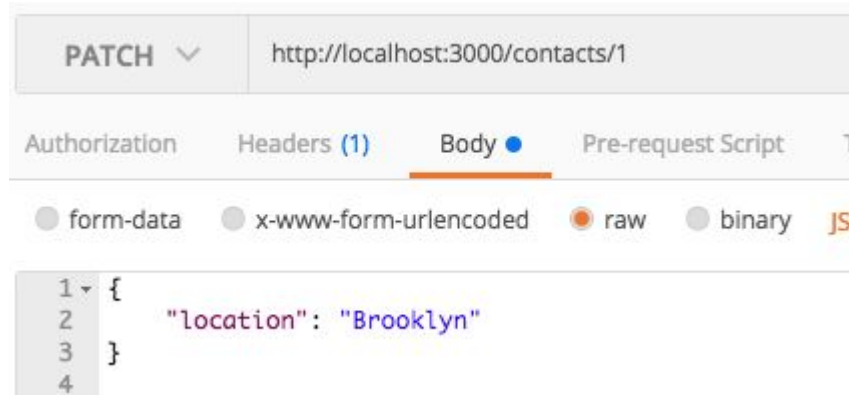
- **DELETE**
- Remove um dado passado na URI

The screenshot shows a REST client interface with the following elements:

- Method:** DELETE (selected from a dropdown)
- URI:** `http://devmedianotesapi.azurewebsites.net/api/notes/91`
- Params:** A button to manage query parameters.
- Send:** A blue button to execute the request.
- Save:** A button to save the request.
- Tabs:** Authorization, Headers (1), Body (selected), Pre-request Script, Tests.
- Body Type:** radio buttons for form-data, x-www-form-urlencoded, raw (selected), and binary.
- Content Type:** JSON (application/json) (selected from a dropdown).
- Body Editor:** A text area with a line number '1' and a cursor at the start.
- Code:** A button in the top right corner.

Verbos HTTP

- **PATCH**
- Envia apenas o que precisa ser alterado, sem a necessidade de precisar enviar todos os dados.



Exemplo de Envio

Nome:

Email:

Telefone:

Enviar

Exemplo de Envio

Nome:

Email:

Telefone:

Atualizar

Exemplo de Envio


Nome:

Email:

Telefone:

Atualizar

Exemplo de Envio

| Nome | Email | Telefone | Ação |
|-----------|----------------------------|---------------|---|
| K Almeida | kelson.almeida@iesp.edu.br | (83) 88888888 |  |

↓
DELETE
Verb

Códigos de Status HTTP

- A cada response, o protocolo HTTP nos retorna um código de “status” referente a cada tipo de retorno.
- Mas, prof. Esses códigos são aleatórios?
 - Não... São numerações pré-definidas e separadas por classes.



404. That's an error.

The requested URL /does_not_exist was not found on this server. That's all we know.



Códigos de Status HTTP

- Estão divididos em 5 classes:
 - **100s:** A solicitação iniciada pelo cliente HTTP continua...
 - **200s:** O pedido feito no *request* foi recebido, compreendido e processado pelo servidor.
 - **300s:** Redirecionamento. Um novo recurso foi substituído pelo recurso solicitado.
 - **400s:** Erros. Houve problema com o pedido.
 - **500s:** A solicitação foi aceita, mas aconteceu algum erro no servidor que impede o processamento completo da solicitação

HTTP STATUS CODES

2xx Success

200 Success / OK

3xx Redirection

301 Permanent Redirect

302 Temporary Redirect

304 Not Modified

4xx Client Error

401 Unauthorized Error

403 Forbidden

404 Not Found

405 Method Not Allowed

5xx Server Error

501 Not Implemented

502 Bad Gateway

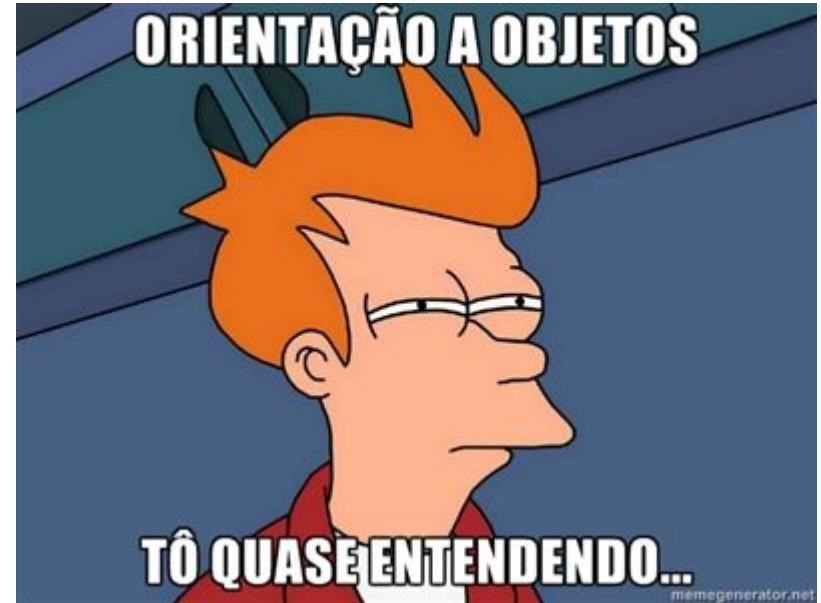
503 Service Unavailable

504 Gateway Timeout

 uniesp

Orientação a Objetos


- Orientação a Objetos é um paradigma de programação que se baseia na ideia de que tudo no mundo é um objeto que pode ser modelado e interagir com outros objetos para realizar tarefas específicas.



Orientação a Objetos

- Um **objeto** é uma instância de uma **classe**... Êpa? É o que prof?
- Vamos com calma!
- Uma “classe” é a estrutura de dados que define as propriedades e comportamentos de um objeto...
- hum...

class UrsinhoPooh



Propriedades: constructor(preguica, cor, camisaVermelha, gostaDeMel) {
 this.preguica = preguica;
 this.camisaVermelha = camisaVermelha;
 this.gostaDeMel = gostaDeMel;
}

The diagram illustrates the relationship between a class and an object. A blue arrow points from the text 'class UrsinhoPooh' to the illustration of Winnie the Pooh. Another blue arrow points from the 'Propriedades:' label to the constructor code block.

Orientação a Objetos

- Um **objeto** é uma instância de uma **classe**... Êpa? É o que prof?
- Vamos com calma!
- Uma “classe” é a estrutura de dados que define as propriedades e comportamentos de um objeto...
- hum...



class UrsinhoPooh

Comportamentos:

```
ficarDePreguicinha() {  
    console.log("zzzZZZzzzZZzz");  
}
```

Orientação a Objetos

- Tá, prof... Mas vimos a classe e seus comportamentos e atributos.. Mas e o objeto em si?
- Lembram que eu falei que ele era uma instância de uma classe?

Objeto instanciado.

- `const ursinhoPooh = new UrsinhoPooh(true, amarelo, true);`

class UrsinhoPooh



Propriedades:

```
constructor(preguica, cor, camisaVermelha,
gostaDeMel) {
    this.preguica = preguica;
    this.camisaVermelha = camisaVermelha;
    this.gostaDeMel = gostaDeMel;
}
```


Orientação a Objetos

- Então, como vimos, as classes são utilizadas para definir objetos e encapsular suas propriedades e comportamentos.
- Opa! Encapsular??
- Sim, agora vamos tentar entender três conceitos básicos de OO:
Encapsulamento, Herança e Polimorfismo

javascript

```
class Carro {  
  constructor(marca, modelo, ano) {  
    this.marca = marca;  
    this.modelo = modelo;  
    this.ano = ano;  
  }  
  
  getInfo() {  
    return `${this.marca} ${this.modelo} ${this.ano}`;  
  }  
  
  acelerar() {  
    console.log("Acelerando...");  
  }  
}
```

Orientação a Objetos

- **Encapsulamento:** Se refere à ideia de que as propriedades e comportamentos de um objeto devem ser mantidos privados e acessíveis apenas por meio de métodos públicos, evitando assim que outras partes do código possam modificar o objeto de maneira inesperada.

```
javascript
function Pessoa(nome, idade) {
  var nome = nome;
  var idade = idade;

  this.getNome = function() {
    return nome;
  };

  this.setNome = function(novoNome) {
    nome = novoNome;
  };

  this.getIdade = function() {
    return idade;
  };

  this.setIdade = function(novaIdade) {
    idade = novaIdade;
  };
}

var pessoa1 = new Pessoa("João", 30);

console.log(pessoa1.getNome()); // "João"
console.log(pessoa1.getIdade()); // 30

pessoa1.setNome("Maria");
pessoa1.setIdade(25);

console.log(pessoa1.getNome()); // "Maria"
console.log(pessoa1.getIdade()); // 25
```

Orientação a Objetos

- **Herança:** Permite que uma classe herde as propriedades e comportamentos de outras classes. Isso pode ajudar a evitar a repetição de código e aumentar a eficiência do desenvolvimento de software.

```
javascript

// Classe Animal
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(this.name + ' faz um barulho.');
```

```
}

// Classe Cachorro que herda de Animal
class Cachorro extends Animal {
  constructor(name) {
    super(name);
  }

  speak() {
    console.log(this.name + ' late.');
```

```
}

// Instância de Cachorro
const cachorro = new Cachorro('Rex');
cachorro.speak(); // Output: Rex late.
```

Orientação a Objetos

- **Polimorfismo:** Capacidade de objetos de diferentes classes responderem ao mesmo método de maneira diferente. Isso permite que o código seja mais flexível e reutilizável, pois os objetos podem ser tratados como se fossem do mesmo tipo, independente de suas diferenças.

```
CSS

function somar(a, b) {
  return a + b;
}

function somar(a, b, c) {
  return a + b + c;
}

console.log(somar(2, 3)); // saída: 5
console.log(somar(2, 3, 4)); // saída: 9
```

```
javascript

class Animal {
  fazerSom() {
    console.log("Som genérico de animal.");
  }
}

class Cachorro extends Animal {
  fazerSom() {
    console.log("Au au!");
  }
}

class Gato extends Animal {
  fazerSom() {
    console.log("Miau!");
  }
}

let animais = [new Cachorro(), new Gato(), new Animal()];

animais.forEach(animal => animal.fazerSom());
```

Orientação a Objetos

- No exemplo ao lado temos duas funções “somar”, mas com parâmetros diferentes. A **primeira tem dois parâmetros, já a segunda possui três parâmetros.**
- Esse é um exemplo de Polimorfismo de Sobrecarga, onde duas funções diferentes **têm o mesmo nome, mas comportamentos diferentes, dependendo dos parâmetros passados.**

CSS

```
function somar(a, b) {  
    return a + b;  
}
```

```
function somar(a, b, c) {  
    return a + b + c;  
}
```

```
console.log(somar(2, 3)); // saída: 5
```

```
console.log(somar(2, 3, 4)); // saída: 9
```

Orientação a Objetos

- Já neste outro exemplo, temos uma classe “Animal”, e duas subclasses “Cachorro” e “Gato”, cada uma com um método chamado “fazerSom”.
- Quando criamos um array chamando cada um dos objetos, será chamado cada método responsável pelos diferentes animais.
- Isso é um exemplo de polimorfismo de substituição, onde diferentes objetos têm o mesmo método, mas comportamentos diferentes.

javascript

```
class Animal {  
  fazerSom() {  
    console.log("Som genérico de animal.");  
  }  
}  
  
class Cachorro extends Animal {  
  fazerSom() {  
    console.log("Au au!");  
  }  
}  
  
class Gato extends Animal {  
  fazerSom() {  
    console.log("Miau!");  
  }  
}  
  
let animais = [new Cachorro(), new Gato(), new Animal()];  
  
animais.forEach(animais => animais.fazerSom());
```

Vamos praticar?

- Faça uma classe, em JS, “Pessoa” que possua as propriedades: nome, idade e profissao.
- Esta classe vai possuir o método “exibirInfo()” que vai printar:
 - Nome: ..., Idade: ..., Profissao: ...
- Fora da classe crie um objeto Pessoa em uma variável.
- Chame o método exibirInfo() através do seu novo objeto.

