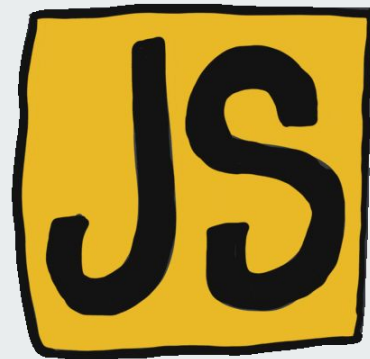




JAVASCRIPT - Objetos e Estruturas de Dados





Objetos

- Objetos em JavaScript, assim como em outras linguagens orientadas a objetos, definem um conjunto de informações (**propriedades**) e um conjunto de ações (**métodos**) que representam algum conceito importante dentro do programa
- Por exemplo, podemos definir um objeto para representar um **carro**. Esse objeto pode ter propriedades como **nome**, **modelo**, **peso** e **cor**. Além disso, ele pode ter métodos que representam ações como **ligar()**, **dirigir()**, **frear()** e **parar()**

myScript.js

```
// Declarando e definindo um objeto chamado car  
const car = {type:"Fiat", model:"500", color:"white"};
```

Objetos

- Objetos em JavaScript, assim como em outras linguagens orientadas a objetos, definem um conjunto de informações (**propriedades**) e um conjunto de ações (**métodos**) que representam algum conceito importante dentro do programa
- Por exemplo, podemos definir um objeto para representar um **carro**. Esse objeto pode ter propriedades como **nome**, **modelo**, **peso** e **cor**. Além disso, ele pode ter métodos que representam ações como **ligar()**, **dirigir()**, **frear()** e **parar()**

myScript.js

```
/* Declarando e definindo um
objeto chamado person */
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50
};
```

myScript.js

```
// Declarando e definindo um objeto chamado car
const car = {type:"Fiat", model:"500", color:"white"};
```



Objetos: métodos

- Métodos em um objeto definem as ações que podem ser executadas por ele
- Os métodos são definidos em um objeto como funções

myScript.js

```
/* O objeto person agora possui um método, chamado
fullName, que retorna o nome completo */
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```



Objetos: acessando propriedades e métodos

- É possível acessar propriedades de um objeto de duas maneiras:
 - `objectName.propertyName`
 - `objectName["propertyName"]`
- Os métodos podem ser acessados como a seguir:
 - `objectName.methodName()`



Datas

- O JavaScript nos permite manipular datas a partir de um tipo específico de objeto para essa finalidade: **Date**
- Por padrão, o JavaScript usará o fuso horário do navegador e exibirá a data completa como uma string, exemplo:
 - `Thu Nov 11 2023 15:14:49 GMT-0300 (Horário Padrão de Brasília)`
- Objetos datas são criados com o construtor **`new Date()`**

myScript.js

```
const d = new Date();
```



Arrays

- Um array é uma variável especial que pode conter mais de um valor:

myScript.js

```
const cars = ["Ford", "Hyundai", "BMW"];
```

- Para saber o tamanho de um array, basta acessar a propriedade **length**: **cars.length**
- Para acessar um elemento do array, assim como em outras linguagens, devemos referenciá-lo pelo seu índice: **cars[i]**
- Para alterar um elemento no array deve-se atribuir o novo valor no índice desejado:

myScript.js

```
cars[0] = "Fiat";
```



Arrays: métodos

- Existem alguns métodos para manipular arrays, como:

Método	Descrição
pop()	remove o último elemento do array
push()	insere um novo elemento ao final do array
shift()	remove o primeiro elemento e altera os índices dos demais
unshift()	adiciona um elemento no início do array e altera os índices
concat()	concatena arrays existentes criando um novo array
splice()	adiciona novos itens em um array, definindo onde serão inseridos e quantos itens serão removidos
slice()	recorta uma parte de um array criando um novo array



Arrays: métodos

- Por exemplo, para remover o último elemento de um array de carros:

myScript.js

```
const cars = ["Ford", "Hyundai", "BMW"];  
cars.pop(); // Remove o elemento "BMW"
```

- Para inserir um novo elemento ao final do array:

myScript.js

```
cars.push("Honda");
```



Arrays: métodos

- O método `slice()`:

myScript.js

```
const cars = ["Ford", "Hyundai", "BMW", "Honda"];  
const cars2 = cars.slice(1, 3);  
console.log(cars2); // Saída: Array ["Hyundai", "BMW"]
```

- ❑ **Sintaxe:** `slice(start, end)` onde `start` corresponde à posição do elemento inicial e `end` corresponde à posição do elemento final (não incluído).



Arrays: métodos básicos de iteração

- Métodos de iteração de arrays operam sobre cada item do array, como:
 - **forEach()**

myScript.js

```
const numbers = [45, 4, 9, 16, 25];  
let txt = "";  
numbers.forEach(myFunction);  
  
function myFunction(value, index, array) {  
    txt += value + "<br>";  
}
```



Arrays: métodos básicos de iteração

- Também é possível iterar um array com a estrutura **for .. of**

myScript.js

```
const letters = ["a", "b", "c"];

for (const x of letters) {
  // bloco de código a ser executado
}
```



Sets

- Um conjunto (**Set**) JavaScript é uma coleção de valores únicos
- Cada valor só pode ocorrer uma vez no conjunto
- Para criar um conjunto deve-se utilizar o construtor: **new Set()**
- Pode-se utilizar o método **add()** para adicionar valores no conjunto

myScript.js

```
// Create um conjunto e iniciando os valores no construtor
const letters = new Set(["a", "b", "c"]);

// Create um conjunto vazio
const letters2 = new Set();

// adicionando valores ao conjunto
letters2.add("a");
letters2.add("b");
letters2.add("c");
```



Sets: iteração com forEach()

- O método **forEach()** invoca uma função para cada elemento do conjunto:

myScript.js

```
// Criando o conjunto
const letters = new Set(["a", "b", "c"]);

// Listando todos os elementos
let text = "";
letters.forEach(function(value) {
  text += value;
})
```

Maps

- Um **Map** JavaScript é uma coleção de pares chave-valor, onde as chaves podem ser de qualquer tipo
- Para criar um Map deve-se utilizar o construtor: **new Map()**
- Pode-se utilizar o método **set()** para adicionar valores no Map

myScript.js

```
// Criando um Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
```

myScript.js

```
// Criando um Map
const fruits = new Map();

// Definindo valores ao Map
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);
```



Maps: iteração com forEach()

- O método **forEach()** invoca uma função para cada chave-valor do Map:

myScript.js

```
// Lista de todas as entradas
let text = "";
fruits.forEach (function(value, key) {
    text += key + ' = ' + value;
})
```



Referências Importantes

MDN – Objetos em JavaScript

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object

MDN – Arrays

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array

MDN – Métodos de array (**pop**, **push**, **splice**, etc.)

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array#métodos

MDN – Set

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Set

MDN – Map

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Map



Exercícios

1. Crie um objeto `livro` com as propriedades `titulo` (string), `autor` (string) e `paginas` (número), e inclua um método `resumo()` que retorne uma frase no formato:

"[**Título**] foi escrito por [**Autor**] e possui [**X**] páginas."

2. Dado o array `const nums = [1, 2, 3, 4, 5]`:
 - Remova o primeiro elemento usando um método de array.
 - Remova o último elemento usando outro método.
 - Adicione o número `0` no início.
 - Adicione o número `6` no fim.
 - Ao final, exiba o array resultante no console.



Exercícios

3. Crie um array de strings com nomes de frutas, por exemplo `["maçã", "banana", "laranja"]`. Utilize `forEach()` para imprimir no console cada fruta junto com seu índice, no formato:

"Fruta **[índice]: [nomeDaFruta]**"

4. A partir do array `const valores = [1,2,2,3,4,4,5]`; , crie um **Set** que elimine os duplicados e, em seguida, converta-o de volta para um array. Exiba o resultado.
5. Crie um **Map** que associe cada dia da semana a um número de horas trabalhadas (por ex.: `"segunda" → 8`, `"terça" → 6`, ...). Depois, use `forEach()` do Map para imprimir no console:

"[dia]: [horas]"