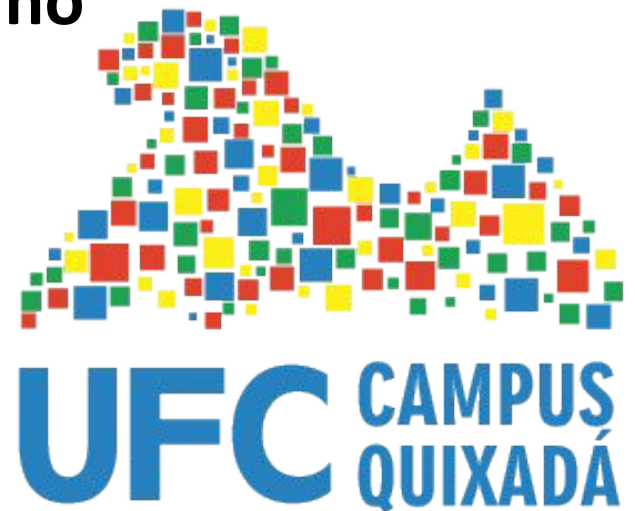


Introdução ao JavaScript

Prof. Sidartha Carvalho



AULA 1/4

JS: introdução

- Trio dos desenvolvedores Web:
 - HTML
 - Para definir o conteúdo das páginas web
 - CSS
 - Para especificar o design das páginas web
 - JavaScript (JS)
 - Para fornecer “comportamento” às páginas web
- Agora vamos focar no JavaScript...
 - Não é usado somente em páginas Web
 - Várias aplicações desktop e servidores também usam JS
 - Por exemplo: NodeJS (servidor), MongoDB, CouchDB, etc

JS: introdução

- JavaScript não é Java...
 - Porém, quando foi criada, o Java estava no auge e tudo que remetia ao Java era considerado “coisa boa”... Daí, foi uma oportunidade de tirar proveito da popularidade do Java para lançar uma nova linguagem.
- JavaScript
 - Criada por Brendan Eich em 1995 e se tornou um padrão em 1997
 - ECMAScript é o nome oficial da linguagem
- O que posso fazer com JS no contexto da web?
 - Adicionar/alterar componentes HTML/CSS e seu conteúdo
 - Adicionar/alterar estilos CSS
 - Adicionar comportamento e acionar eventos
 - Etc...

JS: alterando HTML

```
<!DOCTYPE html>
<html>
<body>

<h2>O que posso fazer com JavaScript?</h2>

<p id="meup">JavaScript pode alterar o conteúdo de um elemento HTML.</p>

<button type="button" onclick='document.getElementById("meup").innerHTML =
  "Novo texto usando JavaScript!'">Click Me!</button>

</body>
</html>
```

JS: alterando HTML

```
<!DOCTYPE html>
<html>
<body>

<p>JavaScript pode alterar os atributos dos elementos HTML.</p>

<p>Vamos alterar o endereço de uma imagem de forma dinâmica.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Li
gar lâmpada</button>



<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">A
pagar a lâmpada</button>

</body>
</html>
```

JS: alterando CSS

```
<!DOCTYPE html>
<html>
<body>


<p id="demo">JavaScript pode alterar o estilo de um elemento HTML.
</p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Clique aqui!</button>

</body>
</html>
```

JS: <script>: alert()

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Aula de JS</title>
  <script>
    alert("Olá, Mundo!");
  </script>
</head>
<body>
  <h1>JavaScript</h1>
  <h2>Linguagem de programação</h2>
</body>
</html>
```



A tag <script> é usada para inserir código JS, pode ser inserida no <head>, no <body> ou em arquivo específico.

Externalizando...

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Aula de JS</title>
  </head>
  <body>
    <h1>JavaScript</h1>
    <h2>Linguagem de Programação</h2>
    <script src="js/hello.js"></script>
  </body>
</html>
```

js/hello.js

```
alert("Olá, Mundo!");
```



```
<html>

<head>
  <title>Calculadora</title>
</head>

<body>
  <form method="POST">
    <input type="text" name="n1" id="n1" />
    <input type="text" name="n2" id="n2" />
    <button type="submit" id="botao">OK</button>
  </form>
  <p id="resultado"></p>
  <script>
    function exibe_resultado() {
      res.textContent = parseFloat(n1.value) + parseFloat(n2.value);
      return false;
    }
    var n1 = document.getElementById('n1');
    var n2 = document.getElementById('n2');
    var botao = document.getElementById('botao');
    var res = document.getElementById('resultado');
    botao.onclick = exibe_resultado;
  </script>
</body>

</html>
```

JS: console.log()

- Função usada para imprimir dados no console
 - Usado para depuração, acompanhamento da execução, erros, informações importantes, etc...
- Dentro de tags <script>, incluir a chamada:

```
console.log('teste')
```

- Para visualizar, abrir o modo desenvolvedor no navegador:
 - Chrome: Control + shift + c
 - Firefox: Control + shift + k

JS: comentários

```
// Comentário em uma única linha

/* Este é um comentário JavaScript em
   várias linhas ....
   ....
   ....
*/
```

JS: tipos de dados

- Tipagem dinâmica, o tipo é atribuído a partir do valor da variável
- Dados primitivos
 - Number
 - String
 - Boolean
 - Undefined
- Dados complexos
 - Object
 - Null
 - Function

JS: expressões

- JS avalia expressões da esquerda para a direita
 - `var x = 16 + 4 + "Volvo";`
 - `= 20Volvo`
 - Para evitar possíveis problemas, use sempre os parênteses.
 - `var x = (16 + 4) + "Volvo";`
 - `= 20Volvo`
 - `var x = "Volvo" + 16 + 4;`
 - `= Volvo164`

JS: tipos de dados

- `var x;` `//` agora x é do tipo `undefined`
- `x = 5;` `//` agora x é do tipo `Number`
- `x = "John";` `//` agora x é do tipo `String`

JS: tipos de dados

- Conversão de string para número (defina a base):
 - `parseInt("123", 10)`
 - 123
 - `parseInt("010", 10)`
 - 10
 - `parseInt("11", 2)`
 - 3

JS: String

- `"hello".length`
 - 5
- Strings são objetos:
 - `"hello".charAt(0)`
 - h
 - `"hello, world".replace("hello", "goodbye")`
 - goodbye, world
 - `"hello".toUpperCase()`
 - HELLO

Strings

String	<code>length</code>	Returns the number of characters in a string
	<code>concat()</code>	Joins two or more strings
	<code>indexOf()</code>	Returns the position of the first occurrence of a specified string value in a string
	<code>lastIndexOf()</code>	Returns the position of the last occurrence of a specified string value, searching backward from the specified position in a string
	<code>match()</code>	Searches for a specified string value in a string
	<code>replace()</code>	Replaces some characters with others in a string
	<code>slice()</code>	Extracts a part of a string and returns the extracted part in a new string
	<code>split()</code>	Splits a string into an array of strings
	<code>substring()</code>	Extracts the characters in a string between two specified indexes
	<code>toLowerCase()</code>	Displays a string in lowercase letters
	<code>toUpperCase()</code>	Displays a string in uppercase letters

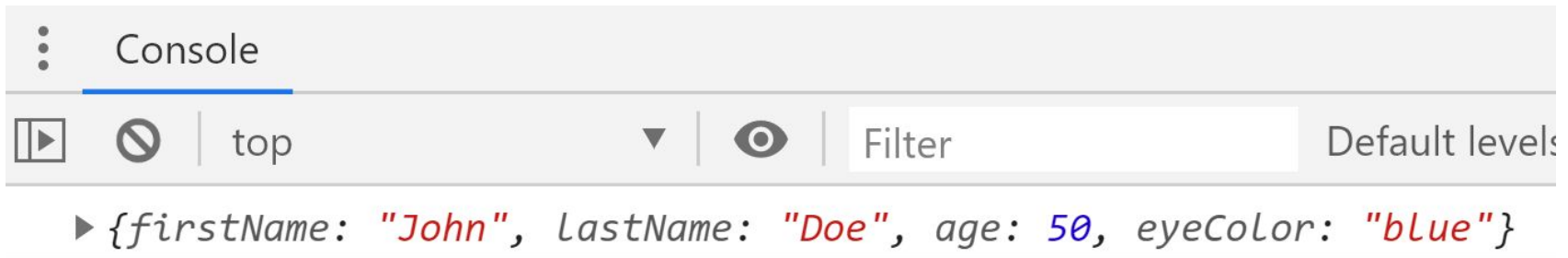
Resumo:
Introdução JS, usos, tipos de dados e
expressões

JS: Arrays

- `var cars = ["Saab", "Volvo", "BMW"];`
- `let cars = ["Saab", "Volvo", "BMW"];`

JS: Objects

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
    console.log(person);
```



JS: variáveis

- Declaradas com a palavra chave var:
 - let a;
 - let name = "simon";
- Declarar sem atribuir valor: undefined
- Se esquecer a palavra var, a variável é global.
 - Não faça isso, por favor.

```
var mensagem = "Olá mundo";  
console.log(mensagem);
```

JS: conversão de tipos

- `let textoInteiro = "10";`
- `let inteiro = parseInt(textoInteiro);`

- `let textoFloat = "10.22";`
- `let float = parseFloat(textoFloat);`

- `let milNumber = 1000;`
- `let milString = milNumber.toFixed(2);` // recebe o retorno da função
- `console.log(milString);` // imprime a string "1000.00"

JS: operadores

- Numéricos: +, -, *, / e %
- Atribuição encurtada: =, +=, -=, *=, /=, %=
- Incremento/decremento: a++, ++a, b--, --b
 - Se usado como operador prefixado (++x), retorna o valor de seu operando após a adição. Se usado como operador pósfixado (x++), retorna o valor de seu operando antes da adição.
 - Se x é 3, então ++x define x como 4 e retorna 4, enquanto x++ retorna 3 e, somente então, define x como 4.

JS: operadores

- Concatenação de string:
 - "hello" + " world"
 - hello world
- Coerção de tipos:
 - "3" + 4 + 5
 - 345
 - 3 + 4 + "5"
 - 75
- Adicionar uma string vazia a alguma coisa, converte tudo para string

JS: comparativos

- Para números e strings: `<`, `>`, `<=` e `>=`
- Igualdade: `==` e `!=`
 - Faz conversão de tipos se necessário
 - `"dog" == "dog"`
 - `true`
 - `1 == true`
 - `true`
- Identidade `===` e `!==` (compara o valor e o tipo do obj):
 - Não faz conversão de tipos
 - Se forem de tipos diferentes, o resultado será falso
 - `1 === true`
 - `false`
 - `true === true`
 - `true`

JS: if

```
var name = "kittens";  
if (name == "puppies") {  
    name += "!";  
} else if (name == "kittens") {  
    name += "!!";  
} else {  
    name = "!" + name;  
}  
  
name == "kittens!!"
```

JS: while e do..while

```
while (true) {  
    // an infinite loop!  
}  
do {  
    var input = get_input();  
} while (inputIsValid(input))
```

JS: for

```
for (var i = 0; i < 5; i++) {  
    // Will execute 5 times  
}
```

```
for (const num of numerosMegaSena)  
{  
    console.log(num);  
}
```

JS: switch

```
switch (action) {  
  case 'draw':  
    drawit();  
    break;  
  case 'eat':  
    eatit();  
    break;  
  default:  
    donothing();  
}
```

Expressões são permitidas
Comparações usam ===

```
switch (1 + 3):  
  case 2 + 2:  
    yay();  
    break;  
  default:  
    neverhappens();  
}
```

JS: &&

&& e || só executam o segundo operando, dependendo do resultado do primeiro.

Útil para checagem de objetos antes de acessar seus atributos:

```
var name = o && o.getName();
```

- Se o for true, retorna o.getName
- Se o for false, retorna false

```
1  a1 = true  && true      // t && t retorna true
2  a2 = true  && false     // t && f retorna false
3  a3 = false && true      // f && t retorna false
4  a4 = false && (3 == 4)  // f && f retorna false
5  a5 = 'Cat' && 'Dog'     // t && t retorna "Dog"
6  a6 = false && 'Cat'     // f && t retorna false
7  a7 = 'Cat' && false     // t && f retorna false
8  a8 = ''    && false     // f && f retorna ""
9  a9 = false && ''        // f && t retorna false
```

- Exemplos de valores false: null; NaN; 0; string vazia (""); undefined.

```
1  o1 = true  || true      // t || t retorna true
2  o2 = false || true      // f || t retorna true
3  o3 = true  || false     // t || f retorna true
4  o4 = false || (3 == 4)  // f || f retorna false
5  o5 = 'Cat' || 'Dog'     // t || t retorna "Cat"
6  o6 = false || 'Cat'     // f || t retorna "Cat"
7  o7 = 'Cat' || false     // t || f retorna "Cat"
8  o8 = ''      || false   // f || f retorna false
9  o9 = false  || ''       // f || f retorna ""
```


JS: operador ternário

```
var allowed = (age > 18) ? "yes" : "no";
```

JS: exceções

```
try {  
    // Statements in which  
    // exceptions might be thrown  
} catch(error) {  
    // Statements that execute  
    // in the event of an exception  
} finally {  
    // Statements that execute  
    // afterward either way  
}
```

```
throw new Error("An error!");  
throw "Another error!";
```

- JavaScript Object Notation.
 - Usado para serializar objetos em formato legível ao ser humano.

```
{ "Books":  
  [  
    { "ISBN":"ISBN-0-13-713526-2",  
      "Price":85,  
      "Edition":3,  
      "Title":"A First Course in Database Systems",  
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},  
                   {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }  
    ,  
    { "ISBN":"ISBN-0-13-815504-6",  
      "Price":100,  
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",  
      "Title":"Database Systems:The Complete Book",  
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},  
                   {"First_Name":"Jeffrey", "Last_Name":"Ullman"},  
                   {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }  
  ]  
}
```

1

- Valores base
 - Número, string, booleano, null
 - Objetos { }
 - Conjuntos de pares chave-valor
 - Arrays []

```
{ "alunos" : [  
    { "nome": "João", "notas": [ 8, 9, 7 ] },  
    { "nome": "Maria", "notas": [ 8, 10, 7 ] },  
    { "nome": "Pedro", "notas": [ 10, 10, 9 ] }  
]  
}
```

JS: criação de objetos

- `var obj = new Object();`

ou

- `var obj = {};`
- Equivalentes. A segunda opção chama-se sintaxe literal de objeto e é mais conveniente.

JS: acesso aos atributos

- `obj.name = "Simon"`
 - `var name = obj.name;`

ou

- `obj["name"] = "Simon";`
 - `var name = obj["name"];`
- Equivalentes.
 - O segundo usa strings, podendo ser decidido em tempo de execução e usado para palavras reservadas.

JS: sintaxe de objetos

```
var obj = {  
  name: "Carrot",  
  "for": "Max",  
  details: {  
    color: "orange",  
    size: 12  
  }  
}
```

```
obj.details.color  
= orange  
  
obj["details"]["size"]  
= 12
```

JS: iterar entre objetos

- Pode-se iterar pelas chaves de um objeto:

```
var obj = { 'name': 'Simon', 'age': 25 };  
for (var attr in obj) {  
    print(attr + ' = ' + obj[attr]);  
}
```


JS: arrays

- Tipo especial de objeto onde as chaves são números e não strings, como o caso anterior.

```
var a = new Array();  
a[0] = "dog";  
a[1] = "cat";  
a[2] = "hen";  
a.Length
```

```
var a = ['dog', 'cat', 'hen']  
a[0]  
a[1]  
a[2]
```

```
var palavras = ["UFC", "Ensino"];  
palavras.push("Inovação"); //add nova palavra ao array
```

JS: arrays

```
var a = ["dog", "cat", "hen"];  
a[100] = "fox";  
a.length  
101
```

```
typeof(a[90])  
Undefined
```

```
Append Seguro (não irá gerar undefined):  
a[a.length] = item;
```

JS: iteração em arrays

```
for (var i = 0; i < a.length; i++) {  
    // Do something with a[i]  
}  
for (var item in a) {  
    // Do something with item  
}  
  
["dog", "cat", "hen"].forEach(  
    function (currentValue, index, array) {  
        // Do something with currentValue or array[index]  
    });
```

JS: arrays

Array	<code>length</code>	Sets or returns the number of members in an array
	<code>concat()</code>	Joins two or more arrays and returns the result
	<code>join()</code>	Puts all the members into a string, separated by the specified delimiter
	<code>pop()</code>	Removes and returns the last element of an array
	<code>push()</code>	Adds one or more members to the end of an array and returns the new length
	<code>reverse()</code>	Reverses the order of the members in an array
	<code>shift()</code>	Removes and returns the first member of an array
	<code>slice()</code>	Returns selected members from an existing array
	<code>sort()</code>	Sorts the members of an array
	<code>splice()</code>	Removes and adds new members to an array
	<code>unshift()</code>	Adds one or more members to the beginning of an array and returns the new length

JS: Funções de Manipulação de Arrays

- Map
 - Usado para criar um novo array a partir de um array existente, aplicando uma mesma função a cada um dos elementos do array inicial.
- Filter
 - Aplica uma instrução condicional a um array. Se essa condição for verdadeira, o elemento é colocado no array de resultado.
- Reduce
 - Reduz um array de valores a um único valor. Para obter o valor de resultado, ele executa uma função de redução em cada elemento do array.

JS: Funções de Manipulação de Arrays

- Map

```
const numbers = [1, 2, 3, 4];  
  
const doubled = numbers.map(item => item * 2);  
  
console.log(doubled); // o resultado é [2, 4, 6, 8]
```

JS: Funções de Manipulação de Arrays

- Filter

```
const numbers = [1, 2, 3, 4];  
  
const evens = numbers.filter(item => item % 2 === 0);  
  
console.log(evens); // o resultado é [2, 4]
```

JS: Funções de Manipulação de Arrays

- Reduce

```
myArr.reduce(callback[, valorInicial])
```

O argumento `callback` é uma função que será chamada uma vez para cada item do array. Essa função recebe quatro argumentos, mas, em geral, apenas os dois primeiros são utilizados.

- *acumulador* - o valor retornado da iteração anterior
- *valorAtual* - o item atual no array
- *índice* - o índice do item atual
- *array* - o array original para o qual o método `reduce` é chamado
- O argumento `valorInicial` é opcional. Se for fornecido, será usado como o valor inicial do acumulador na primeira chamada da função de `callback`.

JS: Funções de Manipulação de Arrays

- Reduce

```
const numbers = [1, 2, 3, 4];

const sum = numbers.reduce(function (result, item) {
  return result + item;
}, 0);

console.log(sum); // o resultado é 10 - 1+2+3+4
```

- Pode ser reescrito usando arrow functions

```
const numbers = [1, 2, 3, 4];

const sum = numbers.reduce((result, item) => result + item, 0);

console.log(sum); // o resultado é 10 - 1+2+3+4
```

JS: Destructuring arrays

- Forma resumida de acessar elementos de um array
 - Ao invés de usar
 - `const winner = scores[0]`
 - `const secondwinner = scores[1]`
- Pode-se utilizar
 - `const [gold, silver] = scores;`
 - `const [gold, silver, ...outros] = scores;`

```
const scores = [90, 85, 80, 75, 70];
```

```
console.log(gold); // Output: 90
```

```
console.log(silver); // Output: 85
```

```
console.log(outros); // Output: [80, 75, 70]
```

JS: Destructuring arrays

- Para valores de um objeto, pode fazer:
 - `const nome = user.nome`
 - `const idade = user.idade`
- Forma resumida
 - `const {nome, idade} = user`
 - Irá linkar os atributos com os nomes das variáveis
- Se quiser mudar o nome da variável:
 - `const {nome: primeiroNome} = user`
 - Pega o `user.nome` e atribui para `primeiroNome`

JS: Destructuring arrays

- Se alguns atributos estiverem presente somente em alguns objetos, é possível definir um valor default
 - `const {nome, idade, filhos = 0}`
 - Então, se o objeto não tiver a propriedade filhos, irá receber o valor default, filhos = 0.
- Destructuring os parâmetros de entrada de uma função também é possível, evitando passar todo o objeto.
 - `function fullName({firstName, lastName}){.....},`
 - Evita ter que passar todos o objeto user e depois fazer a coleta dos dados (user.firstName e user.lastName)

```
function fullName({firstName, lastName}){  
  console.log(firstName)  
  console.log(lastName)  
}
```

```
user = {firstName : 'Siddhartha', middleName: 'Azevedo', lastName : 'Carvalho'}  
fullName(user)
```

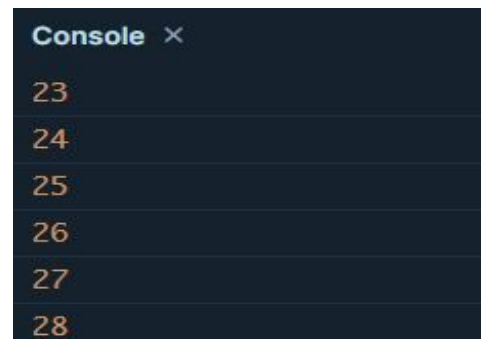
Resumo:

arrays, objetos, variáveis, conversão de tipos, operadores e comparativos, JSON.

EXERCÍCIO 1

- 1. Crie um script que instancia um array com 30 posições e itera sobre ele, salvando em cada elemento o valor da posição dele no array somado de 23. Após isso, imprima no console todos os elementos do array.

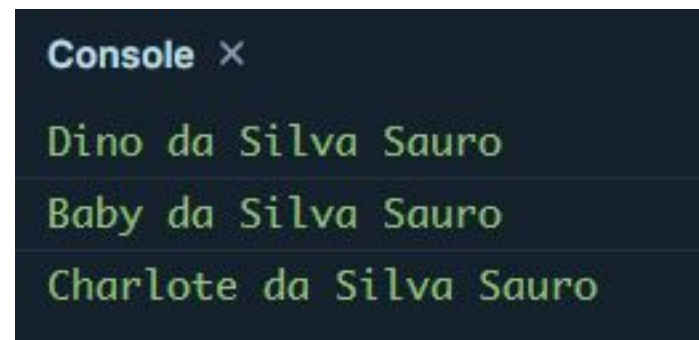
- Deve imprimir do 23 ao 53.



```
Console x
23
24
25
26
27
28
```

- 2. Dado o array “let nomes = ['Dino', 'Baby', 'Charlotte’]” percorra todos os elementos e adicione o sobrenome “ da Silva Sauro” usando a função Map.

- Resultado:



```
Console x
Dino da Silva Sauro
Baby da Silva Sauro
Charlotte da Silva Sauro
```

EXERCÍCIO 2

- Você está desenvolvendo um aplicativo para uma loja online. Um dos requisitos é exibir o preço com desconto de cada produto listado no inventário. A lista de produtos está disponível como um array de objetos, onde cada objeto possui os atributos nome e preço.
- Dado o array de produtos abaixo, escreva um código JavaScript que utilize o método **map** para criar um novo array onde cada produto possui um novo atributo `precoComDesconto` (10% de desconto aplicado). O array resultante deve manter os outros atributos inalterados.

```
const produtos = [  
  { nome: "Camiseta", preco: 50 },  
  { nome: "Calça", preco: 100 },  
  { nome: "Tênis", preco: 200 },  
];
```

EXERCÍCIO 3

- Você está criando uma funcionalidade para um sistema de biblioteca.
- É necessário listar apenas os livros que estão disponíveis para empréstimo.
- Os livros têm as informações título, autor e disponível.
- Com base no array de livros abaixo, escreva um código JavaScript que utilize o método **filter** para retornar um novo array contendo apenas os livros com o atributo disponível igual a true.

```
const livros = [  
  { titulo: "JavaScript para Iniciantes", autor: "João Silva", disponivel: true  
},  
  { titulo: "CSS Avançado", autor: "Maria Oliveira", disponivel: false },  
  { titulo: "React Rápido", autor: "Ana Souza", disponivel: true },  
];
```


EXERCÍCIO 4

- Você está desenvolvendo uma funcionalidade de relatório financeiro para um sistema de vendas. É necessário calcular o valor total das vendas realizadas em um determinado mês.
- Dado o array de vendas abaixo, escreva um código JavaScript que utilize o método **reduce** para calcular e retornar o valor total das vendas.

```
const vendas = [  
  { produto: "Notebook", preco: 2500, quantidade: 2 },  
  { produto: "Smartphone", preco: 1500, quantidade: 3 },  
  { produto: "Teclado", preco: 200, quantidade: 5 },  
];
```