# JavaScript

# Programa do curso

- **Aula 1: introdução e objetos**
- Aula 2: JavaScript integrado a HTML
- Aula 3: DOM, seletores e elementos
- Aula 4: Eventos
- Aula 5: Formulários
- Aula 6: Ajax
- Aula 7: Exercício Integrador

# 1.
# O que é JavaScript?

# Breve história

- A Netscape queria uma web mais dinâmica com scripts que pudessem ser executados no lado do *client* (navegador).
- Em 1990, a Netscape contratou Brendan Eich para criar uma linguagem que fizesse isso.
- Em 1996 a Netscape submeteu o Javascript para a Ecma internacional para padronizar a linguagem.
- Assim nasceu o termo ECMAScript, nome oficial da linguagem.

# O que é JavaScript?

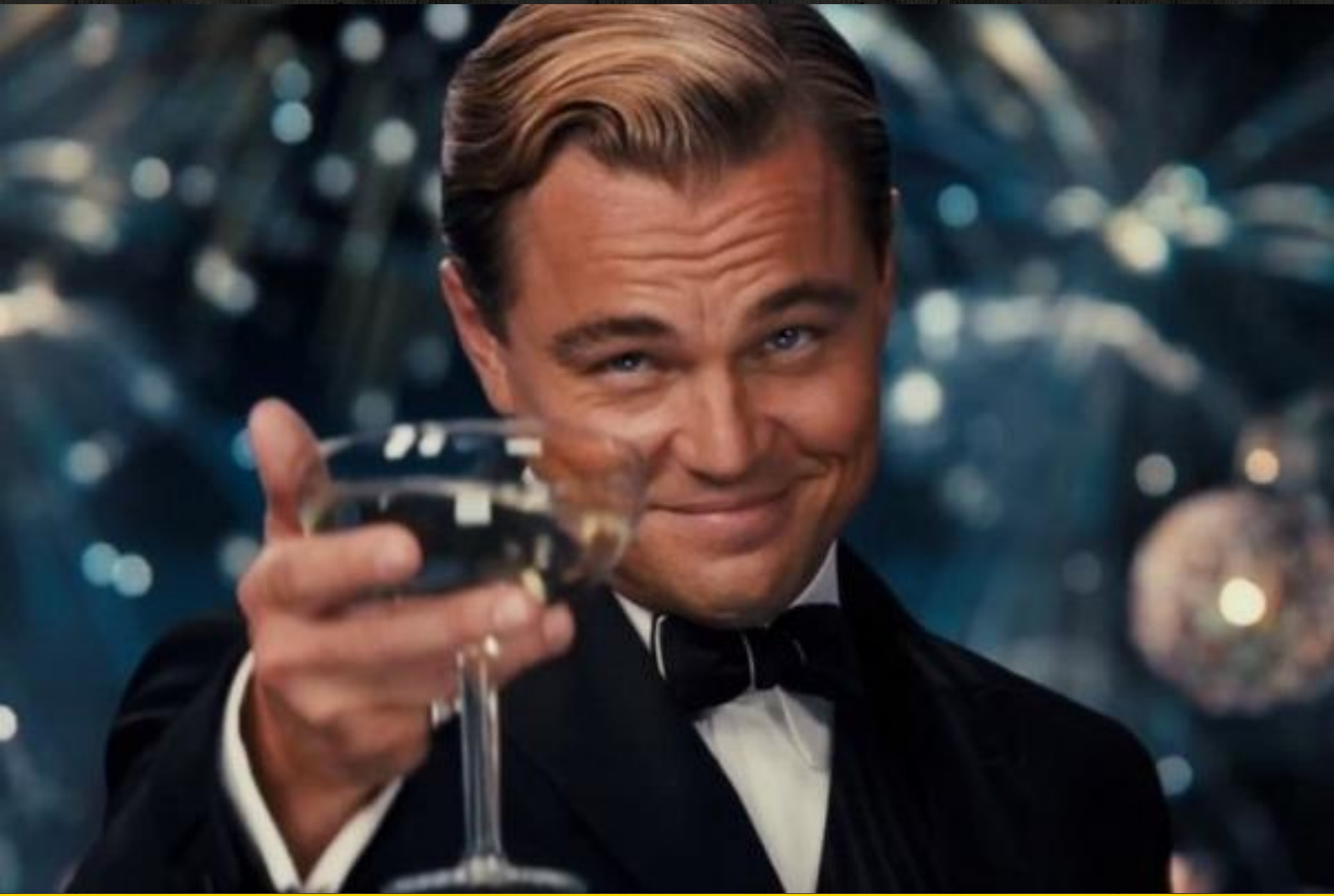
JavaScript é uma linguagem de programação voltada para a web.

- Interpretada
- Multiparadigma
  - Orientada a objetos
  - Funcional
  - Imperativa
- ECMAScript

# Para que serve o JS?

Como é uma linguagem de programação, o JS também pode servir para gerenciar a lógica do back-end, para isso utilizamos o Node.js.

Além disso, os navegadores trazem consigo um interpretador de JS que permite executar JS no cliente, tornando as páginas interativas.



# Sintaxe

# Variáveis

**LET**

```
let nome = ‘Neo’;
let idade = 33;
```

# Tipos básicos

| | |
|---|---|
| ‘pepe’ | string |
| 10.7 | number |
| true | boolean |
| { } | object |
| [ ] | array |
| function() {} | function |
| null | null |
| undefined | undefined |

# 2. Operadores

# Operador de atribuição

**let nome = ‘Neo’;**

Utilizamos o símbolo “=”

```
let nome = ‘Pepe’;
let idade = 30;
```

# Operadores aritméticos

| | |
|---|---|
| let num1 = 5; | |
| let num2 = 3; | |
| num1 + num2 | adição, subtração |
| num1 * num2 | multiplicação, divisão |
| num1++ | incremento |
| num1-- | decremento |

# Operadores de atribuição e aritméticos

| | |
|---|---|
| let numero = 10 | |
| numero **+=** 2 | 12 |
| numero **-=** 2 | 8 |
| numero ***=** 2 | 20 |
| numero **/=** 2 | 5 |

# Operadores de comparação simples e estrita

## Simples

| | |
|---|---|
| == | igualdade de valor |
| != | valor diferente |
| <= | menor e igual |
| >= | maior ou igual |

## Estrita

| | |
|---|---|
| === | igualdade de valor e tipo |
| !== | diferente valor e tipo |

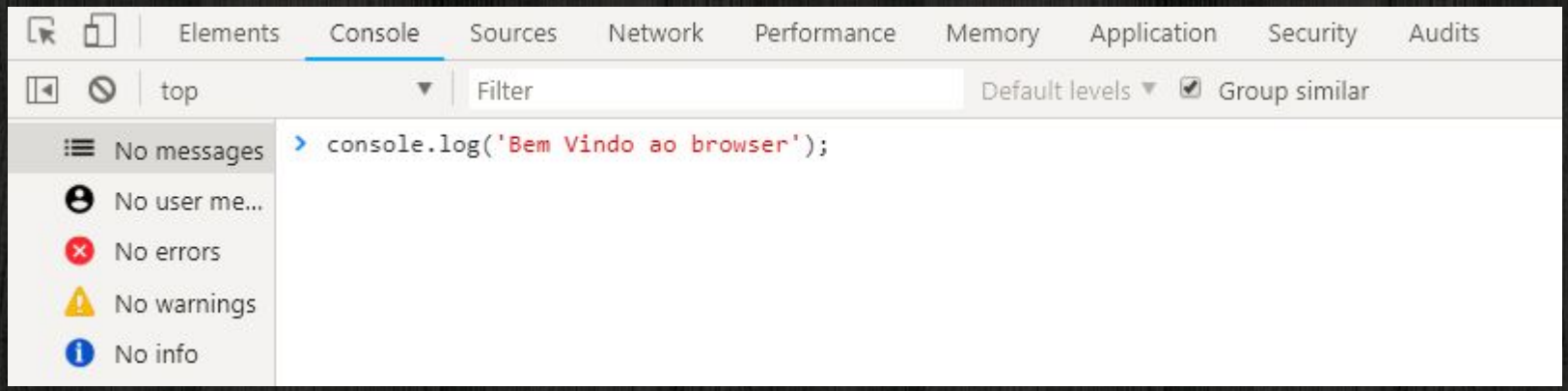
# Operadores lógicos

| | | |
|---|---|---|
| && | Operador and (e) | a && b |
| \|\| | Operador or (ou) | a \|\| b |
| ! | Operador de negação | !false = true |

# A linha de comando

Os navegadores trazem uma linha de comando incorporada para programar JavaScript.

Normalmente usamos F12 para abri-la e começar a escrever o código.



# if

## Exemplo

```
let idade = 20;

if (idade >= 18) {
    console.log("Acesso liberado”);
}
```

Isso imprimirá na linha de comando:

“Acesso liberado”

# Truthy e Falsy

Um valor **truthy** é um valor que, quando avaliado como boolean, se transforma em verdadeiro.

| | |
|---|---|
| if (false){ // Isto NÃO seria executado } | falsy |
| if (null) | falsy |
| if (undefined) | falsy |
| if (0) | falsy |
| if (NaN) | falsy |
| if ("") | falsy |
| if ("foo") | truthy |

# If ternário

**expressao ? resultado True : resultado False**

| | |
|---|---|
| expressao | Qualquer expressão **booleana** |
| resultado true | Expressão retornada se expressao for **true** |
| resultado false | Expressão retornada se expressao for **false** |

## Exemplo

```
let eMembro = true;
"A Mensalidade é:  " + (eMembro ? "$2.00" : "$10.00");
"A mensalidade é: $2.00"
```

# Switch

```
let fruta = "mamão";

switch (fruta) {
        case "cereja":
                console.log("A cereja do bolo");
        break;

        case "abacaxi":
                console.log("Me ajuda a descascar esse abacaxi?");
        break;

        case "mamão":
                console.log("É mamão com açúcar!");
        break;

        default:
                console.log("É outra fruta");
}
```

# For

```
for (inicio; condicao; incremento) {
        // fazer isso enquanto a condição for verdadeira
}
```

## Exemplo

```
for (let i = 0; i < 4; i++) {
        console.log("Olá, " + i);
}
```

Isso imprimirá na linha de comando:

"Olá, 0"

"Olá, 1"

"Olá, 2"

"Olá, 3"

# For - break

```
for (let i = 0; i < 4; i++) {
        console.log("Olá " + i);
        if(i === 1){
                break; // interrompe o loop FOR
        }
}
```

Isso imprimirá na linha de comando:

```
"Olá, 0"
"Olá, 1"
```

# While

```
while (condição) {
        // executar enquanto a condição for verdadeira
}
```

## Exemplo

```
let a = 0;
while (a < 3) {
        console.log("Olá");
        a++; // Sempre chegar à condição de corte
}
```

Isso imprimirá na linha de comando:

```
"Olá"
"Olá"
"Olá"
```

# 1. Funções

# Definição de funções

Uma função é um bloco de códigos designado para realizar uma tarefa.

```
function multiplicar(n1, n2) {
    return n1 * n2;  // Retorna a multiplicação de n1 com n2
}
```

Uma função é executada somente quando é invocada:

```
multiplicar(2, 5); // Isso retornará 10
```

# Scope

O scope de uma variável é o contexto em que ela é visível.

```
function myFunction() {
    var a = 4;
    return a * a;
}
```

Neste exemplo, **a** é uma variável **local.**

```
var a = 4;
function myFunction() {
    return a * a;
}
```

Neste exemplo, a é uma variável **global**.

# Funções aninhadas

Podemos ter uma função dentro de outra função.

Uma função aninhada é ‘visível’ apenas dentro da função pai.

```
function circunferencia (raio)
{
    function diametro()   // função aninhada
    {
        return 2 * raio;
     }

    return Math.PI * diametro(); // invocamos a função
}
```

# Vamos ver este exemplo

```
function a(){
  console.log( 'a vem primeiro');
}
function b(){
  console.log( 'b vem depois' );
}
a();
b();
```

O resultado seria:

```
a vem primeiro
b vem depois
```

# O que acontece neste exemplo?

```
function a(){
  setTimeout( function(){
     console.log( 'a vem primeiro');
  }, 1000 );
}

function b(){
  console.log( 'b vem depois' );
}
a();
b();
```

Ficaria assim:

```
b vem depois
a vem primeiro
```

# *Callbacks*

# Callbacks

Callback é uma função passada como parâmetro a outra função, que será executada após o término da primeira.

JavaScript é uma linguagem assíncrona. Isso significa que é possível executar uma invocação sem saber quando ela termina. Para poder administrar essa situação, utilizamos o padrão de design **callback.**

Os callbacks podem ser usados de várias formas. No próximo exemplo, utilizamos um callback com uma função anônima.

```
function a(callback){
        setTimeout( function(){
                console.log( 'a vem primeiro');
                callback();
        }, 3000 );
}

function b(){
        console.log( 'b vem depois' );
}

a( b );
```

```
a vem primeiro
b vem depois
```

# Vamos praticar!

Prática 2 - **Funções**

# 2. Arrays

# Métodos de arrays - forEach

Exemplo:

```
let numeros = [1, 5, 7];
let resultado = numeros.forEach(function(value, index) {
  console.log("No índice: " + index + " está o valor: " + value);
});
```

Resultado:

```
No índice: 0 está o valor: 1
No índice: 1 está o valor: 5
No índice: 2 está o valor: 7
```

# Métodos de arrays - map

O **map** é usado para modificar **cada um** dos itens de um array utilizando uma função determinada.

```
let numeros = [1, 5, 7];
let resultado = numeros.map(function(numero) {
        return numero * 2;
});
```

O resultado é um novo array.

```
[2, 10, 14]
```

# Métodos de arrays - filter

O **filter** é usado para filtrar alguns elementos de um array.

```
let numeros = [13, 18, 20];
let resultado = numeros.filter(function(numero) {
        return (numero >= 18);
});
```

O resultado é um novo array.

```
[18, 20]
```

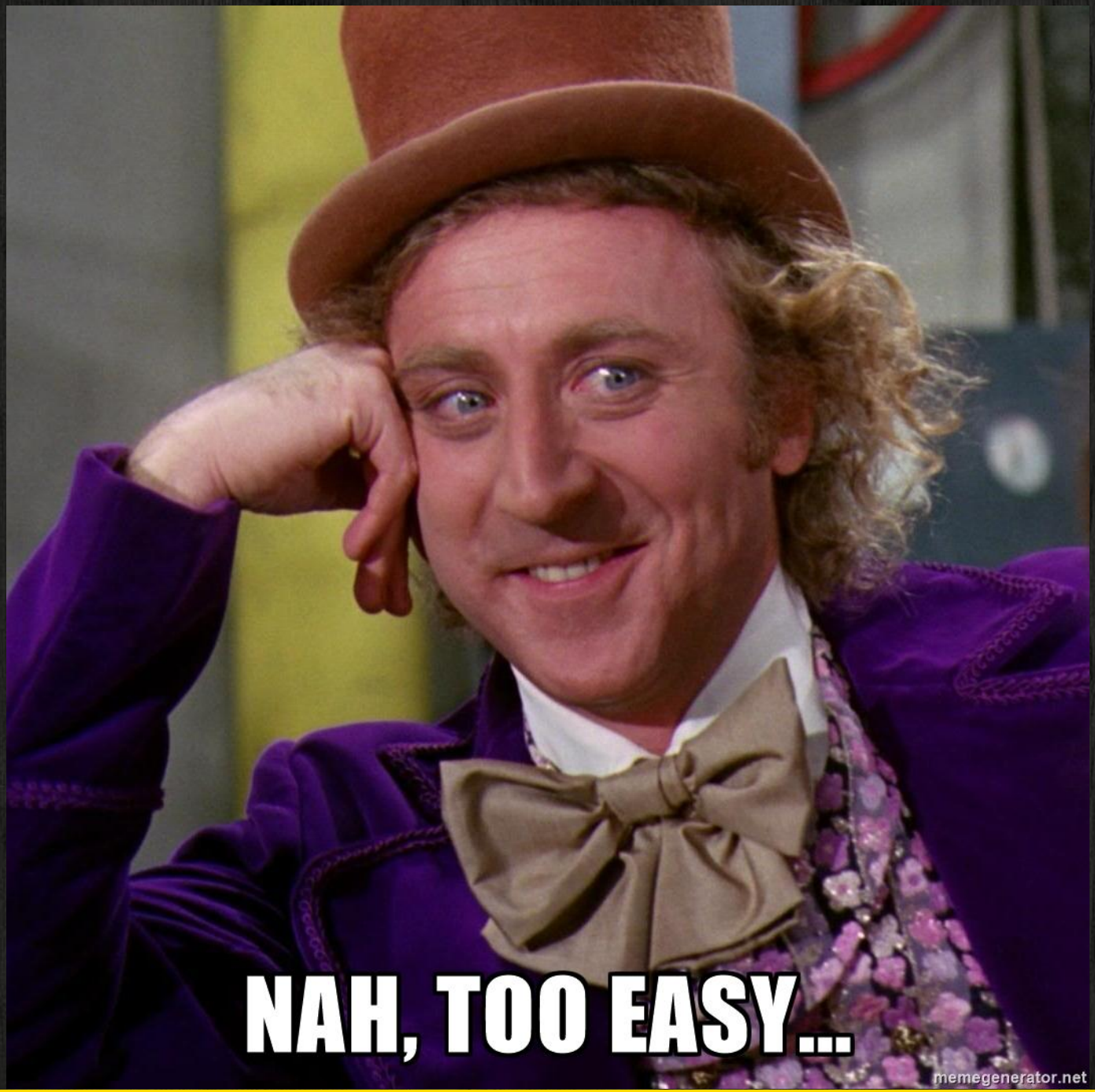
# Métodos de arrays - reduce

O **reduce** é usado para reduzir um array a um valor único utilizando uma função determinada.

```
let numeros = [1, 5, 7];
let resultado = numeros.reduce(function(total, numero) {
        return total + numero;
}, 0);
```

O resultado é um valor único.

```
13
```


NAH, TOO EASY...
memegenerator.net

# Vamos praticar!

Prática 2 - **Arrays**

# 3. Objeto literal

# Definição de objetos

Um **objeto** é como uma váriavel mas com a capacidade de guardar uma chave e valor.

```
var carro = {
    marca: "Chevrolet",
    modelo: "Corsa",
    quilometragem: 65000,
    cor: "Branco",
    donos: ["Luciana", "João", "Zezinho"]
};
```

Os objetos têm **propriedades** usadas para descrevê-lo.

Elas podem ser de vários tipos (number, string, array, object, etc.)

# Propriedades de objetos

É possível acessar uma **propriedade** de um objeto da seguinte forma:

- carro.marca; // "Chevrolet”

# Propriedades de objetos

Estabelecer o valor de uma propriedade:

```
carro.cor = "Vermelho";
```

É possível adicionar novas propriedades a um objeto:

```
carro.velocidadeMax = 200;
```

# Métodos de objetos

Também existem **métodos** para interagir com os objetos.

Um método é uma propriedade do objeto com uma **função** atribuída.

```
var carro = {
        marca: "Chevrolet",
        ligar: function(){
                console.log("vrum vrum");
        },
        modelo: "Corsa"
};
carro.ligar();
```

# Métodos de objetos e parâmetros

Como são funções, os métodos também podem receber parâmetros.

```
var carro = {
        marca: "Chevrolet",
        motorista: function(nome){
                console.log(nome+" é o motorista");
        },
        modelo: "Corsa"
};
carro.motorista("Cachorro");
```

“Cachorro está dirigindo”

# Vamos praticar!

Prática 2 - **Objeto Literal**

*Obrigado!*

# Perguntas?