

Desenvolvimento BACK-END I

Professor: Douglas Legramante

E-mail: douglas.legramante@ifro.edu.br



INSTITUTO FEDERAL
Rondônia
Campus Vilhena



JS

JavaScript

Aula 04 - Objetos

O que é um objeto?

Um objeto é uma coleção de dados e/ou funcionalidades relacionadas (que geralmente consistem em diversas variáveis e funções — que são chamadas de propriedades e métodos quando estão dentro de objetos).

Em JavaScript tudo (ou quase tudo) é um objeto, ou age como um objeto.

Qualquer valor que não seja de um tipo primitivo (uma string, um número, um booleano, um símbolo, null ou undefined) é um **objeto**.

Objeto

Um objeto é composto de vários membros, cada um com um nome e um valor. As propriedades de um objeto são representadas por pares de nome/valor. Cada par nome/valor deve ser separado por uma vírgula e o nome e valor, em cada caso, separados por dois pontos. A sintaxe sempre segue esse padrão:

```
var nomeDoObjeto = {  
    nomeMembro1: valorMembro1,  
    nomeMembro2: valorMembro2,  
    nomeMembro3: valorMembro3,  
};
```

Objeto literal

Um objeto literal é uma forma conveniente de criar e inicializar objetos. Ele permite definir um objeto de forma direta, sem a necessidade de criar uma classe ou usar o operador **new**.

```
var produto = {  
  id: 9,  
  nome: "Cafeteira Elétrica",  
  valor: 99.00  
};
```

A importância do objeto literal

Utilizando variáveis para armazenar os dados de um produto:

3 variáveis

```
var id = 9;  
var nome = "Cafeteira Elétrica";  
var valor = 99.00;  
  
console.log(id); // 9  
console.log(nome); // Cafeteira Elétrica  
console.log(valor); // 99.00
```

Utilizamos uma variável para armazenar cada dado do produto. Mas será essa a melhor maneira?

A importância do objeto literal

E se no mesmo sistema precisarmos armazenar o id e o nome tanto de um produto como de um cliente?

dados de um produto

```
var id = 9;  
var nome = "Cafeteira Elétrica";  
var valor = 99.00;  
  
console.log(id); // 9  
console.log(nome); // Cafeteira Elétrica  
console.log(valor); // 99.00
```

dados de um cliente

```
var id = 40;  
var nome = "Jorge Mendes";  
var telefone = "(21) 9999-9999";  
  
console.log(id); // 40  
console.log(nome); // Jorge Mendes  
console.log(telefone); // (21) 9999-9999
```

A importância do objeto literal

Essa situação gera um problema, pois não conseguimos identificar com clareza a qual domínio (contexto) uma variável pertence.

dados de um produto

```
var id = 9;  
var nome = "Cafeteira Elétrica";  
var valor = 99.00;  
  
console.log(id); // 9  
console.log(nome); // Cafeteira Elétrica  
console.log(valor); // 99.00
```

dados de um cliente

```
var id = 40;  
var nome = "Jorge Mendes";  
var telefone = "(21) 9999-9999";  
  
console.log(id); // 40  
console.log(nome); // Jorge Mendes  
console.log(telefone); // (21) 9999-9999
```


A importância do objeto literal

Resolvemos este problema utilizando o **objeto literal**, que nos permite agrupar os dados de uma maneira eficiente.

```
var produto = {  
  id: 9,  
  nome: "Cafeteira Elétrica",  
  valor: 99.00  
};  
  
console.log(produto.id); // 9  
console.log(produto.nome); // Cafeteira Elétrica  
console.log(produto.valor); // 99.00
```

Agrupando dados em
um objeto literal

A importância do objeto literal

Os dados contidos dentro do objeto literal pertencem a um domínio (contexto) específico.

```
var produto = {  
  id: 9,  
  nome: "Cafeteira Elétrica",  
  valor: 99.00  
};  
  
console.log(produto.id); // 9  
console.log(produto.nome); // Cafeteira Elétrica  
console.log(produto.valor); // 99.00
```

A propriedade nome está relacionada ao objeto literal produto

A importância do objeto literal

Dessa forma não temos problema, pois cada propriedade é relacionada a um objeto literal.

dados de um produto

```
var produto = {  
  id: 9,  
  nome: "Cafeteira Elétrica",  
  valor: 99.00  
};
```

```
console.log(produto.id); // 9  
console.log(produto.nome); // Cafeteira Elétrica  
console.log(produto.valor); // 99.00
```

dados de um cliente

```
var cliente = {  
  id: 40,  
  nome: "Jorge Mendes",  
  telefone: "(21) 9999-9999"  
}
```

```
console.log(cliente.id); // 40  
console.log(cliente.nome); // Jorge Mendes  
console.log(cliente.telefone); // (21) 9999-9999
```

Criando objetos

Notação Literal:

```
const carro = {  
  
}
```

Sintaxe new Object:

```
const carro = new Object()
```

Sintaxe Object.create():

```
const carro = Object.create()
```

Criando objetos com new

O operador **new** cria e inicializa um novo objeto. A palavra-chave **new** deve ser seguida de uma chamada de função (). Uma função usada dessa maneira é chamada de construtora e serve para inicializar um objeto recém-criado. O JavaScript contém construtoras internas para tipos nativos. Por exemplo:

```
var o = new Object(); // Cria um objeto vazio: o mesmo que {}.  
var a = new Array(); // Cria um array vazio: o mesmo que [].  
var d = new Date(); // Cria um objeto Date representando a hora atual  
var r = new RegExp("js"); // Cria um objeto RegExp para comparação de padrões.
```

Criando objetos com new

Além dessas construtoras internas, é comum definir suas próprias funções construtoras para inicializar objetos recém-criados.

```
function Carro(marca, modelo) {  
  this.marca = marca  
  this.modelo = modelo  
}
```

Essa função servirá como um construtor para o objeto.

```
let meuCarro = new Carro('Fiat', 'Pulse')  
meuCarro.marca // 'Fiat'  
meuCarro.modelo // 'Pulse'
```

Os objetos são **sempre** passados como referência.

Dados de Tipos Primitivos são passados como valor e Objetos são passados como referência.

Por Valor: Significa criar uma CÓPIA do valor original.

```
let idade = 36
let minhaIdade = idade
minhaIdade = 37
console.log(idade) //36
```

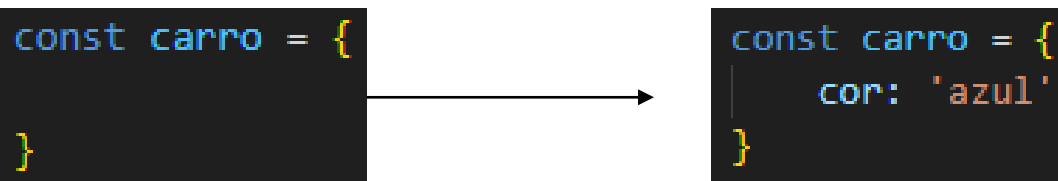
Por Referência: Significa criar um APELIDO (alias) para o valor original.

```
const carro = {
  cor: 'azul'
}
const outroCarro = carro
outroCarro.cor = 'amarelo'
console.log(carro.cor) //'amarelo'
```

Propriedades dos objetos

Os objetos possuem **propriedades**, que são compostas por um nome (ou chave) associado a um valor.

O valor de uma propriedade pode ser de qualquer tipo, o que significa que pode ser um array, uma função e até mesmo um objeto, pois objetos podem ser aninhados em outros objetos.



```
const carro = {  
}
```

```
const carro = {  
  cor: 'azul'  
}
```

Aqui temos um objeto **carro** com uma propriedade chamada **cor**, de valor **azul**.

Chaves (*strings*) com ou sem aspas

Os nomes ou chaves que rotulam as propriedades podem ser qualquer string, mas tome cuidado com caracteres especiais. Caracteres inválidos para nomes de variáveis incluem espaços, hifens e outros caracteres especiais.

```
const carro = {  
  cor: 'azul',  
  'a cor': 'azul'  
}
```

As propriedades
são separadas com
uma vírgula.

Para incluir um caractere inválido para nomes de variáveis na chave da propriedade, temos que colocar o nome/chave entre aspas.

Acessando propriedades de um objeto

Podemos acessar os valores das propriedades de um objeto de duas maneiras:

1. **Notação de colchetes:** utilizada para propriedades com nomes inválidos. O nome/chave (string), sempre entre aspas.

```
console.log(carro['a cor']) //'azul'
```

2. **Notação de ponto:** a chave deve obedecer às regras de nomes válidos (por exemplo, elas não devem conter espaços).

```
console.log(carro.cor) //'azul'
```

Se você
acessar uma
propriedade
inexistente,
obterá um
valor
undefined

```
console.log(carro.marca)  
//undefined
```

Propriedades dos objetos

Objetos podem ter outros objetos aninhados como propriedades.

```
const carro = {  
  marca: {  
    nome: 'Ford'  
  },  
  cor: 'azul'  
}
```

Para acessar o nome da marca usamos:

```
carro.marca.nome
```

Ou:

```
carro['marca']['nome']
```

Propriedades dos objetos

Podemos definir o valor de uma propriedade ao definir o objeto e atualizá-lo em seguida.

```
const carro = {  
  cor: 'azul'  
}  
  
carro.cor = 'amarelo'  
carro['cor'] = 'vermelho'
```

Podemos adicionar novas propriedades a um objeto:

```
carro.modelo = 'Fiesta'  
carro.modelo // 'Fiesta'
```

Podemos excluir uma propriedade dele usando:

```
delete carro.modelo
```

Coleção de objetos

Quando agrupamos vários objetos de um mesmo contexto temos uma coleção de objetos.

Objeto

```
let disciplina = {  
  id: 3,  
  nome: "História",  
  carga_horaria: 160,  
};
```

Coleção de objetos

```
let colecao_disciplinas = [  
  { id: 1, nome: "Português", carga_horaria: 240 }, // índice 0  
  { id: 2, nome: "Matemática", carga_horaria: 220 }, // índice 1  
  { id: 3, nome: "História", carga_horaria: 160 }, // índice 2  
  { id: 4, nome: "Geografia", carga_horaria: 140 }, // índice 3  
  { id: 5, nome: "Química", carga_horaria: 160 }, // índice 4  
  { id: 6, nome: "Física", carga_horaria: 150 }, // índice 5  
  { id: 7, nome: "Inglês", carga_horaria: 120 }, // índice 6  
];
```

Coleção de objetos

Para exibir os dados, precisamos identificar o índice do objeto.

```
let colecao_disciplinas = [  
  { id: 1, nome: "Português", carga_horaria: 240 }, // índice 0  
  { id: 2, nome: "Matemática", carga_horaria: 220 }, // índice 1  
  { id: 3, nome: "História", carga_horaria: 160 }, // índice 2  
  { id: 4, nome: "Geografia", carga_horaria: 140 }, // índice 3  
  { id: 5, nome: "Química", carga_horaria: 160 }, // índice 4  
  { id: 6, nome: "Física", carga_horaria: 150 }, // índice 5  
  { id: 7, nome: "Inglês", carga_horaria: 120 }, // índice 6  
];
```

```
console.log(colecao_disciplinas[2]);  
//{id: 3, nome: 'História', carga_horaria: 160}
```

```
console.log(colecao_disciplinas[6].id); // 7  
console.log(colecao_disciplinas[6].nome); // Inglês  
console.log(colecao_disciplinas[6].carga_horaria); // 120
```

Métodos de objetos

As funções podem ser atribuídas a uma propriedade de função e, nesse caso, são chamadas de **métodos**.

```
const carro = {  
  marca: "Ford",  
  modelo: "Fiesta",  
  ligar: function () {  
    console.log("Ligado");  
  },  
};  
  
carro.ligar();
```

Neste exemplo, a propriedade **ligar** tem uma função atribuída e podemos invocá-la usando a sintaxe de ponto que usamos para propriedades, com os parênteses no final.

Métodos de objetos

Dentro de um método definido usando uma sintaxe `function() {}`, temos acesso à instância do objeto referenciando-a com a palavra **this**.

```
const car = {  
  marca: "Ford",  
  modelo: "Fiesta",  
  ligar: function () {  
    console.log(`${this.marca} ${this.modelo} ligado!`);  
  },  
};  
  
car.ligar();
```

Neste exemplo, temos acesso aos valores das propriedades `marca` e `modelo` usando `this.marca` e `this.modelo`.

Obs.: Não temos acesso ao **this** ao usar uma **arrow function**.

Métodos de objetos

Métodos podem aceitar parâmetros, como funções regulares.

```
const carro = {  
  marca: "Ford",  
  modelo: "Fiesta",  
  irPara: function (destino) {  
    console.log(`Indo para ${destino}`);  
  },  
};  
  
carro.irPara("Roma");
```

Classes

O que são classes? Elas são uma maneira de definir um padrão comum para múltiplos objetos.

Podemos criar uma classe chamada **Pessoa** (observe o **P** maiúsculo, uma convenção ao usar classes), que possui uma propriedade **nome**:

```
class Pessoa {  
    nome;  
}
```

A partir dessa classe, inicializamos um objeto **flavio** assim:

```
const flavio = new Pessoa()
```

flavio é uma instância da classe **Pessoa**. Podemos definir o valor da propriedade **nome**:

```
flavio.nome = 'Flavio'
```

e podemos acessá-lo usando:

```
flavio.nome
```

Classes

As classes podem conter propriedades e métodos.

Os métodos são definidos desta maneira:

```
class Pessoa {  
  nome;  
  saudacao() {  
    return `Olá, meu nome é ${this.nome}`  
  }  
}
```

Podemos invocar métodos em uma instância da classe:

```
const flavio = new Pessoa();  
flavio.nome = 'Flavio';  
console.log(flavio.saudacao());
```

Classes

Existe um método especial chamado `constructor()`, que podemos usar para inicializar as propriedades da classe quando criamos uma instância de objeto.

Funciona assim:

```
class Pessoa {  
    constructor(nome) {  
        this.nome = nome;  
    }  
  
    saudacao() {  
        return `Olá, meu nome é ${this.nome}.`;  
    }  
}
```

Agora, podemos instanciar um novo objeto da classe, passar uma string e, quando chamarmos `saudacao()`, receberemos uma mensagem personalizada:

```
const flavio = new Pessoa("Flavio");  
console.log(flavio.saudacao());
```

Exercícios

1. Crie a variável **apartamento** e atribua a ela um objeto literal com as seguintes propriedades:
 - quartos = 2
 - tipo = “apartamento”
 - endereco = “Avenida Principal, 456 - Centro”
 - andar: 7

Em seguida, exiba no console a seguinte frase, utilizando todas as propriedades da variável `casa`: "Apartamento com 2 quartos, localizado no 7º andar da Av. Principal, 456 - Centro.".

Exercícios

2. Imagine que você está desenvolvendo um sistema para gerenciar produtos em um armazém. Crie uma variável chamada **produtoEmbalado** que execute as seguintes operações:

Atribua um objeto literal à variável com as seguintes propriedades:

- nome: "Laptop HP"
- categoria: "Eletrônicos"
- peso: 1.5
- preco: 3500.00

Exiba no console uma frase que utilize todas as propriedades do objeto, como o exemplo abaixo: "O produto Laptop HP, da categoria Eletrônicos, pesando 1.5 kg, está à venda por R\$ 3500.00."

Exercícios

Modelagem de Classe para Representar Imóveis

3. Crie uma classe chamada **Imovel** com os seguintes atributos:

- quartos:
- tipo:
- endereço:

Crie um método na classe chamado **exibirInformacoes** que retorna uma string com as informações do imóvel.

Em seguida, instancie dois objetos da classe **Imovel** representando uma **casa** e um **apartamento** com as seguintes características:

Casa:

- Quartos: 4
- Tipo: "Casa"
- Endereço: "Rua da Amizade, 789 - Bairro Alegre"

Apartamento:

- Quartos: 2
- Tipo: "Apartamento"
- Endereço: "Avenida da Paz, 123 - Centro"

Chame o método **exibirInformacoes** para cada instância e exiba o resultado no console.

Exercícios

Modelagem de Classe para Representar Veículos

4. Crie uma classe chamada **Veiculo** com os seguintes atributos:
 - marca
 - modelo
 - ano

Crie um método na classe chamado **exibirDetalhes** que retorna uma string com as informações do veículo.

Em seguida, instancie dois objetos da classe **Veiculo** representando um **carro** e uma **motocicleta** com as seguintes características:

Carro:

- Marca: "Toyota"
- Modelo: "Corolla"
- Ano: 2022

Motocicleta:

- Marca: "Honda"
- Modelo: "CBR 600RR"
- Ano: 2021

Chame o método **exibirDetalhes** para cada instância e exiba o resultado no console.

Para saber mais

O básico sobre objetos – Java Script

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/Basics>

Objetos JavaScript

https://www.w3schools.com/js/js_objects.asp

Cursos Gratuitos

Curso JavaScript Curso em Video

<https://www.cursoemvideo.com/curso/javascript/>

Curso JavaScript YouTube

<https://www.youtube.com/watch?v=McKNP3g6VBA>