Objetos em JavaScript

1. Introdução aos Objetos

Em JavaScript, um **objeto** é um valor composto, que permite agrupar múltiplos valores (sejam primitivos ou outros objetos) em uma única estrutura. Esses valores são armazenados como **propriedades**, que possuem **nome** e **valor** associados.

Dessa forma, um objeto pode ser visto como uma **coleção não ordenada de propriedades**, em que cada chave é única.

```
let pessoa = {
  nome: "Ana",
  idade: 25,
  estudante: true
};

console.log(pessoa.nome);  // "Ana"
console.log(pessoa.idade); // 25
```

Objetos funcionam como **mapas de strings para valores**. Contudo, eles vão além disso:

- Todo objeto herda propriedades de outro objeto, chamado de **protótipo**.
- Esse mecanismo é chamado de **herança prototípica**, uma das principais características do JavaScript.
- Objetos são **dinâmicos**, ou seja, é possível adicionar, alterar ou remover propriedades em tempo de execução.

Qualquer valor em JavaScript que **não seja** `string`, `number`, `boolean`, `symbol`, `null` ou `undefined` é considerado um **objeto**.

2. Criando Objetos

Existem diferentes formas de criar objetos em JavaScript.

2.1. Object Literal

O modo mais simples é usando **literais de objeto**, com chaves `{}`:

```
let carro = {
  marca: "Toyota",
  modelo: "Corolla",
  ano: 2020
};

console.log(carro.marca); // "Toyota"
```

Sempre que o literal é avaliado, um **novo objeto distinto** é criado.

2.2. Criando Objetos com `new`

Outra forma é usar o operador `new` junto a uma função construtora:

```
function Pessoa(nome, idade) {
  this.nome = nome;
  this.idade = idade;
}

let joao = new Pessoa("João", 30);
console.log(joao.nome); // "João"
```

Aqui, a função `Pessoa` atua como **construtor**, inicializando o novo objeto.

2.3. Usando `Object.create()`

O método `Object.create()` permite criar um novo objeto a partir de um protótipo específico:

```
let animal = { tipo: "mamífero" };

let gato = Object.create(animal);
gato.nome = "Mimi";

console.log(gato.nome);   // "Mimi"
console.log(gato.tipo);   // "mamífero" (herdado do protótipo)
```

Podemos até criar objetos sem protótipo:

```
let objSemProto = Object.create(null);
console.log(Object.getPrototypeOf(objSemProto)); // null
```

3. Propriedades

Cada propriedade em um objeto possui **nome** (ou chave) e **valor**.

- É possível acessar propriedades de duas formas:
    - Notação de ponto: `obj.propriedade`
    - Notação de colchetes: `obj["propriedade"]`

```
let livro = { titulo: "JS Definitivo", autor: "David Flanagan" };

console.log(livro.titulo);       // "JS Definitivo"
console.log(livro["autor"]);     // "David Flanagan"
```

A notação de colchetes é útil quando a chave da propriedade é dinâmica ou contém caracteres especiais.

Objetos como Arrays Associativos

Objetos JavaScript funcionam como **arrays associativos**, permitindo criar propriedades em tempo de execução:

```
let portfolio = {};
let propriedade = "açãoApple";

portfolio[propriedade] = 5000;
console.log(portfolio["açãoApple"]); // 5000
```

4. Herança de protótipo

Cada objeto possui um **protótipo**, de onde herda propriedades e métodos.

```
let pessoa = { nome: "Carlos" };
console.log(pessoa.toString()); // herdado de Object.prototype
```

A relação entre objetos e seus protótipos forma a chamada **cadeia de protótipos** (*prototype chain*).

Exemplo com herança:

```
let animal = { tipo: "mamífero" };
let cachorro = Object.create(animal);
cachorro.nome = "Rex";

console.log(cachorro.tipo); // "mamífero" (herdado)
```

5. Erros de Acesso a Propriedades

Nem sempre acessar ou definir propriedades funciona como esperado:

- Se a propriedade não existe, o resultado é `undefined`:

```
let obj = {};
console.log(obj.inexistente); // undefined
```

- `null` e `undefined` não possuem propriedades. Tentar acessá-las causa erro:

```
let valor = null;
console.log(valor.prop); // TypeError
```

- O ES2020 trouxe o **operador de encadeamento opcional** (`?.`), que evita erros:

```
let usuario = {};
console.log(usuario.endereco?.rua); // undefined (sem erro)
```

6. Restrições na Atribuição de Propriedades

Nem sempre é possível modificar uma propriedade:

- Se for **somente leitura (read-only)**.
- Se o objeto não for extensível (`Object.preventExtensions()`).
- Se a propriedade for herdada e imutável.

Em **modo estrito**, tentar modificar nesses casos lança um `TypeError`.

```
"use strict";
let obj = {};
Object.defineProperty(obj, "fixo", { value: 10, writable: false });

obj.fixo = 20; // TypeError
```

Conclusão

- Objetos em JavaScript são coleções dinâmicas de propriedades.
- Podem ser criados com literais `{}`, `new`, ou `Object.create()`.
- Permitem acesso por `.` ou `[]`.
- São mutáveis e comparados por referência.
- Possuem herança prototípica via cadeia de protótipos.
- Propriedades podem ser configuradas como **writable, enumerable, configurable**.
- O acesso a propriedades inexistentes retorna `undefined`.