Objetos em JavaScript

Introdução a Objetos

- Um objeto é um **valor composto**: agrega múltiplos valores (primitivos ou outros objetos).
- É uma **coleção não ordenada** de propriedades, cada uma com **nome e valor**.
- Objetos mapeiam **strings** para valores.
- Mas são mais do que mapas: herdam propriedades de outro objeto (protótipo).
- Essa **herança por protótipo** é fundamental em JavaScript.

```
let pessoa = {
  nome: "Maria",
  idade: 30
};

console.log(pessoa.nome); // "Maria"
```

Mais sobre Objetos

- Objetos são **dinâmicos** → podem ganhar ou perder propriedades.
- Qualquer valor em JavaScript que não seja:
  - string, number, boolean, null, undefined ou symbol é um objeto.
- Objetos são **mutáveis** e manipulados por **referência**.
- Nenhum objeto pode ter duas propriedades com o mesmo nome.
- Propriedades podem ser:
  - **próprias (own properties)**
  - ou **herdadas (prototype)**

Atributos de Propriedades

- Por padrão, as propriedades são:
  - **writable** (podem ser modificadas),
  - **enumerable** (visíveis em loops `for...in`),
  - **configurable** (podem ser apagadas ou redefinidas).

```
let obj = { x: 10 };
Object.defineProperty(obj, "x", { writable: false });

obj.x = 20; // não muda em modo normal; erro em strict mode
console.log(obj.x); // 10
```

Criando Objetos

Criando Objetos

Três formas principais:

1. **Object Literal**
2. **Operador `new` com construtores**
3. **Função `Object.create()`**

Object Literal

- Lista de pares `nome: valor` separados por vírgula, dentro de `{}`.
- Cada vez que é avaliado, cria um **novo objeto distinto**.

```
let pessoa = { nome: "Ana", idade: 25 };
console.log(pessoa.nome); // "Ana"
```

- Dentro de loops, gera múltiplos objetos:

```
for (let i = 0; i < 3; i++) {
  let obj = { numero: i };
  console.log(obj);
}
```

Criando Objetos com `new`

- O operador `new` cria e inicializa um novo objeto.
- Usado junto de uma **função construtora**.

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

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

Prototypes

- O objeto criado por `new` herda do protótipo da função.
- `{}` herda de `Object.prototype`.
- `Object.prototype` não tem protótipo.
- Isso forma a **prototype chain**.

```
let obj = {};
console.log(Object.getPrototypeOf(obj) === Object.prototype); // true
```

Criando Objetos com `Object.create()`

- Cria um novo objeto com o protótipo informado.

```
let pai = { saudacao: "Olá" };
let filho = Object.create(pai);

console.log(filho.saudacao); // "Olá" (herdado)
```

- `Object.create(null)` cria objeto sem protótipo.
- Útil para evitar modificações indesejadas em objetos.

Consultando e Definindo Propriedades

Consultando e Definindo Propriedades

- Acesso por `.` → quando o nome é fixo.
- Acesso por `[]` → quando o nome é dinâmico.

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

console.log(carro.marca); // "Toyota"
console.log(carro["modelo"]); // "Corolla"
```

Objetos como Arrays Associativos

- Objetos funcionam como **arrays associativos**.
- Propriedades podem ser criadas dinamicamente.

```
let portifolio = {};
let acao = "AAPL";

portifolio[acao] = 50;
console.log(portifolio.AAPL); // 50
```

- Diferente de linguagens fortemente tipadas, não há número fixo de propriedades.

Inheritance (Herança)

- Objetos possuem propriedades próprias (***own properties***) e herdadas (***prototype properties***).
- Atribuições afetam sempre o **objeto original**.

```
let pai = { cor: "azul" };
let filho = Object.create(pai);

filho.cor = "verde";
console.log(filho.cor); // "verde"
console.log(pai.cor);   // "azul"
```

- Se tentar sobrescrever uma propriedade somente leitura herdada, a atribuição falha.

Property Access Errors

- Se a propriedade não existe → retorna `undefined`.
- `null` e `undefined` não possuem propriedades → erro.

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

let vazio = null;
// console.log(vazio.x); // TypeError
```

Acesso Condicional (ES2020)

↗ Operador `?.` evita erro ao acessar propriedades de `null` ou `undefined`.

```
let usuario = null;
console.log(usuario?.nome); // undefined (sem erro)
```

Restrições em Atribuições

Falhas em `strict mode` ocorrem se:

- Propriedade for **read-only**.
- O objeto for **não extensível**.
- Tentativa de sobrescrever propriedade herdada não configurável.

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

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