Objetos no JavaScript

> *"Objetos em JavaScript, assim como em muitas outras linguagens de programação, podem ser comparados com objetos na vida real. O conceito de objetos em JavaScript pode ser entendido com objetos tangíveis da vida real.*
> *Em JavaScript, um objeto é uma entidade independente, com propriedades e tipos. Compare-o com uma xícara, por exemplo. Uma xícara é um objeto, com propriedades. Uma xícara tem uma cor, uma forma, peso, um material de composição, etc. Da mesma forma, objetos em JavaScript podem ter propriedades, que definem suas características." - MDN Web Docs*

Propriedades de Objetos:

Uma propriedade pode ser dita como uma variável diretamente ligada a um objeto. Essas propriedades funcionam da mesma maneira que as outras variáveis normais, e, portanto, são dos tipos já conhecidos em JavaScript que vimos até agora, essas variáveis são então agregadas a um Objeto e recebem o nome de propriedades. Vamos pensar da seguinte forma:

```
// Temos um carro x
const carro = {}; // Este é nosso carro qualquer.

// Vemos que acima nosso carro ainda está sem algumas informações importantes,
// como: marca, modelo, ano de fabricação...
// Essas informações são o que chamamos de "Propriedades", vamos exemplificar abaixo:

carro.anoFab = 2001;
carro.modelo = "Vectra";
carro.marca = "GM";

// Acima determinamos algumas propriedades que foram colocadas dentro do Objeto 'carro'
// Caso queiramos conferir se realmente os dados do objeto foram alterados podemos
// utilizar um console.log(), como segue:

console.log(carro);

// { anoFab: 2001, modelo: 'Vectra', marca: 'GM' }
```

Propriedades Indefinidas:

As propriedades que não forem definidas ao se passar valores para dentro de um Objeto não serão salvas dentro deste, como segue no exemplo:

```
// Vamos utilizar aqui novamente a estrutura de carros que já temos, portanto:

const carro = {
    anoFab: 2001,
    marca: "GM",
    modelo: "Vectra"
}

// Agora vamos implementar uma propriedade que não receba valor algum

carro.noProperty; // Salvará undefined

// Ao verificicar o Objeto carro com console.log() teremos o seguinte:

console.log(carro);

// { anoFab: 2001, marca: 'GM', modelo: 'Vectra' }
```

Acessando Objetos:

Para acessar propriedades em Objetos do JavaScript o processo é tão fácil quanto salvar as propriedades, todavia, utilizamos de outra estrutura, como segue no exemplo:

```
// Vamos então continuar com o objeto carro que nós ja temos:

const carro = {
  anoFab: 2001,
  marca: "GM",
  modelo: "Vectra"
}

// Para acessar as propriedades de um objeto, podemos utilizar a sintaxe de membro,
// da seguinte forma:
console.log(carro.anoFab); // 2001
console.log(carro.marca); // GM
console.log(carro.modelo); // Vectra

// Também podemos utilizar a sintaxe ['nomeDaPropriedade'], como segue no exemplo:

console.log(carro["anoFab"]); // 2001
console.log(carro["modelo"]); // Vectra
console.log(carro["marca"]); // GM

// Podemos a partir disto montar estruturas de formatações de texto, como segue:

console.log(`Marca: ${carro["marca"]}, modelo: ${carro["modelo"]}, ano: ${carro["anoFab"]}`);
// Esse carro é da marca GM, o modelo é Vectra e seu ano de fabricação é 2001
```

Existem diversos modos de acessar as propriedades de um objeto além desse, este por exemplo é o método mais usual, podemos também salvar esses dados como "carro["marca"]" dentro da `variável marca` e utilizá-lo como variável global, para mais informações sobre métodos de acesso de Objetos, é possível encontrar a documentação em:

> Trabalhando com objetos
>
> A linguagem JavaScript é projetada com base em um simples paradigma orientado a objeto. Um objeto é uma coleção de propriedades, e uma propriedade é uma associação entre um nome (ou chave) e um valor. Um valor de propriedade pode ser uma função, que é então considerada um método do objeto.
>
> https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objetos

---

Iteração de Objetos:

É possível iterar objetos assim como iteramos Arrays. Para isso iremos criar a seguinte função:

```
const carro = {
    anoFab: 2001,
    marca: "GM",
    modelo: "Vectra"
}

// Agora criamos uma função que itera diferentes objetos:
function showProps(obj, objName) {
    let result = ``
    for (let i in obj) {
        // O método hasOwnProperty() retorna um booleano indicando se o objeto possui a
        // propriedade especificada como uma propriedade definida no próprio objeto em questão
        if (obj.hasOwnProperty(i)) {
            result += `${objName}.${i} = ${obj[i]}\n`
        }
    }
    return result
}

// Agora chamamos a função com o nome que demos ao nosso Objeto:
console.log(showProps(carro, 'carro'))
// carro.anoFab = 2001
// carro.marca = GM
// carro.modelo = Vectra
```

Removendo Propriedades de um Objeto:

É também possível remover propriedades de um Objeto de diferentes modos, para isso vamos usar o nosso já conhecido Objeto 'carro':

```
var carro = {
  anoFab: 2021,
  marca: "Hyundai",
  modelo: "HB20"
```

```
}

// Para remover itens de dentro deste Objeto basicamente utilizamos o métood delete,
// como segue:

delete carro.anoFab;

// Agora conferimos se realmente foi deletado o item em questão utilizando console.log()

console.log(carro);

// {marca: "Hyundai", modelo: "HB20"}
```

Comparando as propriedades de um Objeto:

Os operadores de comparação são muito utilizados quando estamos falando de variáveis em JavaScript, o mesmo vale para os objetos, na aferição de igualdade podemos também ter objetos inclusos, como segue no exemplo:

```
// Associando dois Objetos de mesmo valor:

var fruta = {name: "banana"};
var fruta_copia = fruta;

console.log(fruta == fruta_copia); // true
console.log(fruta === fruta_copia); //true
```

Mas fique atento, dois objetos com mesmo valores não terão 'true' como resposta, para exemplificar sugerimos que teste a estrutura abaixo:

```
var fruta = {name: "banana"};
var fruta_copia = {name: "banana"};

// Agora vamos novamente fazer nossas comparações:

console.log(fruta == fruta_copia); // false
console.log(fruta === fruta_copia); // false
```

Atributos internos das propriedades:

Cada propriedade possui quatro atributos internos, sendo eles:

- configurable;
- enumerable;
- value;
- writable;

A propriedade que focaremos nesse momento é a enumerable, porém fique a vontade em se aventurar e aprender sobre as outrar também.

Enumerable (enumerável) é responsável por determinar se uma propriedade está acessível quando iteramos o objeto pelo `for-in` / `for-of`. Por padrão, todas as propriedades do objeto são enumeráveis, sejam criadas por atribuição ou por um inicializador. Por exemplo:

```
var carro = {
  anoFab: 2021,
  marca: "Hyundai",
  modelo: "HB20"
}

carro.preco = 52290;

console.log(carro);
// Retorna: { anoFab: 2021, marca: 'Hyundai', modelo: 'HB20', preco: 52290 }
```

Contudo, caso essa propriedade seja alterada pelo método Object.defineProperty(), não será renderizada, como segue no exemplo:

```
var carro = {
  anoFab: 2021,
  marca: "Hyundai",
  modelo: "HB20"
}

carro.preco = 52290;

Object.defineProperty(carro, 'combustivel', {
  enumerable: false, //true se quiser que apareça
  value: 'Álcool'
})

console.log(carro);

// Retorna: { anoFab: 2021, marca: 'Hyundai', modelo: 'HB20', preco: 52290 }
```

Repare que não será retornado no console pois `enumerable` é `false`.

Função construtora:

Além da maneira convencional que já aprendemos, conseguimos criar objetos através de funções construtoras. Mas o que é uma função construtora e como a uso?

Primeiramente, esse tipo de função é declarada igualmente à função de expressão ou de declaração, sua diferença está em seu caso de uso e no que ela retorna. As funções construtoras devem possuir seu nome com a primeira letra em **maiúsculo**.

Sua sintaxe é:

```
function FuncaoQualquer() { }
```

Ao executarmos a mesma, devemos utilizar o operador `new` anteriormente. Vamos usar a seguinte função como exemplo:

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

let pessoa = new Pessoa('Clóvis', 30, 'Programador');

console.log(pessoa.nome);
// Retorna: Clóvis

console.log(pessoa.idade);
// Retorna: 30

console.log(pessoa.profissao);
// Retorna: Programador
```

Quando a função é executada com o `new`, é feito o seguinte:

- Um novo objeto vazio é criado e atribuído a this;
- O corpo da função é executado. Normalmente ele modifica this, adiciona novas propriedades a ele.
- O valor de this é retornado no console.

Você pode criar vários objetos `pessoa` caso precise, para isso utilize o operador `new` como no exemplo acima.

Além de propriedades conseguimos atribuir um método dentro do objeto, como:

```
function Pessoa(nome, idade, profissao) {
  this.nome = nome;
  this.idade = idade;
  this.profissao = profissao;
  this.mensagem = function() {
    alert('Olá, '+ this.nome)
  }
}

let pessoa = new Pessoa('Clóvis', 30, 'Programador');
pessoa.mensagem();
```

Perceba que atribuímos uma função a `this.mensagem`. Ela é responsável por alertar o usuário com seu nome, nesse caso retornaria '`Olá, Clóvis`'.

Uma outra possibilidade é possuir um objeto dentro de outro, mas como farei isso? Preste atenção no exemplo:

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

let clovis = new Pessoa('Clóvis', 30, 'Programador');

function Cargo(funcionario, salario){
  this.funcionario = funcionario;
  this.salario = salario;
}

let funcionario1 = new Cargo(clovis, 5000);

console.log(funcionario1);
/* Retorna:
  Cargo {funcionario: Pessoa, salario: 5000}
    funcionario: Pessoa
      idade: 30
      nome: "Clóvis"
      profissao: "Programador"
    salario: 5000
*/
```

Usando esse modo, fazemos o seguinte:

- 1 → Criamos a função `Pessoa()` e passamos os parâmetros requisitados.
- 2 → Geramos um objeto pessoa a partir da variável 'clovis', passando os dados necessários entre parâmetros;
- 3 → Criamos a função `Cargo()` e passamos os parâmetros requisitados;
- 4 → Geramos um objeto cargo a partir da variável '`funcionario1`', passando os dados necessários entre parâmetros;
- 5 → Ao passarmos os dados como parâmetros, perceba que inserimos o objeto '`clovis`';
- 6 → Ele mostra no console o resultado dessa variável '`funcionario1`', e conseguimos ver um objeto "dentro" de outro como dito anteriormente;

Criar objetos com funções construtoras é uma boa prática, já que cria moldes definidos para o objeto, por exemplo recebendo atributos e métodos.

Método Object.create():

Como acabamos de aprender, além do modo padrão de definir um objeto podemos defini-lo com uma função construtora. Um outro meio possível é o Object.create(), ele não possui a necessidade de se definir uma função construtora. Por exemplo:

```
// 1:
var Carro = {
  tipo: "SUV",
  // 2:
  categoria: function () {
    console.log(this.tipo);
  }
}

// 3:
var hillux = Object.create(Carro);
// 4:
hillux.categoria(); // Function

// 5:
var celta = Object.create(Carro);
// 6:
celta.tipo = 'Econômico';
// 7:
celta.categoria();
```

- 1 → Criamos um objeto chamado `Carro` e inserimos as propriedades tipo e categoria;

- 2 → Criamos a função responsável por mostar o tipo do carro em questão;
- 3 → Usamos o método responsável por criar um novo objeto carro (`Object.create`(Carro));
- 4 → Retorna '`SUV`' pois o tipo padrão do objeto tem esse valor;
- 5 → Criamos a variável '`celta`' e atribuímos um novo objeto carro;
- 6 → Atribuímos o valor '`Econômico`' à propriedade tipo;
- 7 → Retorna `Econômico`.

Usando this para referências de objetos:

Como já deve ter percebido, usamos a palavra 'this' para as propriedades do Objeto, mas o que é isso? Quando utilizado fora de escopo, `this` referencia o objeto global Window. Já quando é utilizado dentro de um objeto (que é o nosso caso), vai se referir à instância e poderá acessar suas propriedades.

Removendo propriedades:

Possuímos um operador chamado `delete`, ele é responsável por remover propriedades. Por exemplo:

```
let objetoQualquer = {};

objetoQualquer.propriedade1 = 'abc';
objetoQualquer.propriedade2 = 'def';

console.log(objetoQualquer);
// Retorna: { propriedade1: 'abc', propriedade2: 'def' }

delete objetoQualquer.propriedade1;

console.log(objetoQualquer);
// Retorna: { propriedade2: 'def' }
```

Perceba que no primeiro retorno possuímos 2 propriedades e depois que excluímos a propriedade1 nos é retornado apenas a segunda.

Namespaces:

Basicamente, namespaces são objetos do Javascript nomeados no namespace global. Isso se faz necessário para facilitar nosso código e ajudar em manutenções futuras no mesmo. Por exemplo:

```
// Em vez de realizarmos o seguinte script:

var x = 10;
var obj = {};
var string = 'Olá clóvis';
function alertUser() {
  alert('Olá pessoal...')
};

obj.x = x;
obj.string = string;
obj.function = alertUser;

console.log(obj);
// Retorna: { x: 10, string: 'Olá clóvis', function: ƒ }
```

```
// Podemos definir um namespace exclusivo para determinada função e inserir propriedades no mesmo:

obj = {};
obj.x = 10;
obj.string = 'Olá clóvis';
obj.function = function () { alert('Olá pessoal...') }

console.log(obj);
// Retorna: { x: 10, string: 'Olá clóvis', function: ƒ }
```

Assim, melhoramos tanto a nossa escrita quanto o próprio desempenho do sistema, pois não carregará tantas informações globais.

Percorrendo os dados de um objeto sem saber os nomes das propriedades:

Ainda nesse material vimos um pouco de como percorrer dados de um objeto, porém vamos nos aprofundar um pouco mais. De início veremos quatro modos.

Objeto usado nos exemplos:

```
var pessoa = {
  nome: "Clóvis",
  idade: 30,
  vendedor: true
};
```

- 1 → No ECMAScript 5, combinando `Object.Keys()` e `Array.prototype.forEach()`:

```
Object.keys(pessoa).forEach(function (key) {
  console.log(key, pessoa[key]);
});

/* Retorna:
nome Clóvis
idade 30
vendedor true
*/
```

Da primeira forma, utilizamos o `Object.keys`() no nosso objeto para retornar suas propriedades. Logo em seguida usamos um `forEach`() para executar o que está entre chaves para cada elemento da matriz retornada pelo primeiro método. A função recebe uma determinada 'key' e logo em seguida mostra no console sua key (nesse caso a `propriedade` em si) e seu valor (nesse caso `pessoa[key]`).

- 2 → No ECMAScript 6, adicionando o `for-of`:

```
for (const key of Object.keys(pessoa)) {
  console.log(key, pessoa[key]);
};

/* Retorna:
nome Clóvis
idade 30
vendedor true
*/
```

Do segundo jeito, usamos um `for-of`, ele também receberá o nome da propriedade no lugar de 'key'. Então, para todos os elementos, mostrará no console a propriedade (`key`) seguida de seu valor (`pessoa[key]`).

- 3 → No ECMAScript 8, adicionando o `Object.entries()`:

```
Object.entries(pessoa).forEach(
  ([key, value]) => console.log(key, value)
);

/* Retorna:
nome Clóvis
idade 30
vendedor true
*/
```

Já no terceiro modo usamos o Object.entries() juntamente com o forEach(). Com essa combinação, a procura de cada valor do objeto é evitada. Então, buscamos as propriedades e seus respectivos valores e, para cada resultado (forEach), executamos uma Arrow Function (fique tranquilo caso não entenda alguns conceitos neste momento, estamos apenas vendo exemplos) e por fim mostramos ambos no console.

- 4 → `For-of` e `Object.entries()`:

```
for (const [key, value] of Object.entries(pessoa)) {
  console.log(key, value);
};

/* Retorna:
nome Clóvis
```

```
idade 30
vendedor true
*/
```

Na quarta e última possibilidade usamos o for-of juntamente com o Object.entries(). Esse método pode parecer mais simples e mais curto para alguns, pois usamos um conceito chamado '**Atribuição via Desestruturação**', responsável por definir quais elementos devem ser extraídos da variável de origem.

- Exemplo simples de Atribuição via desestruturação:

```
let numeros = [3, 6, 9, 12];
let [num1, num2] = numeros;

console.log(num1);   // Retorna: 3
console.log(num2);   // Retorna: 6
```

Conceitos dos métodos usados:

- Object.keys() → responsável por retornar uma matriz dos nomes das propriedades enumeráveis do objeto passado entre parênteses;
- Object.entries() → retorna uma matriz de [key, value], pares de propriedades com chave de string enumeráveis;
- forEach() → laço responsável por executar determinada lógica para cada elemento da matriz retornada.

As estruturas mais comuns de Objetos estão exemplificadas acima, todavia, caso queira mais conteúdo sobre objetos, sugerimos que veja os seguintes links:

Trabalhando com objetos

A linguagem JavaScript é projetada com base em um simples paradigma orientado a objeto. Um objeto é uma coleção de propriedades, e uma propriedade é uma associação entre um nome (ou chave) e um valor. Um valor de propriedade pode ser uma função, que é então considerada um método do objeto.

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objetos

JavaScript Objects

In real life, a car is an object. A car has properties like weight and color, and methods like start and stop: Object Properties Methods car.name = Fiat car.model = 500 car.weight = 850kg car.color = white car.start() car.drive() car.brake() car.stop() All cars have the same properties, but the property values differ from car to car.

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

Bibliografias:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects;
- https://ricardo-reis.medium.com/propriedades-enumeráveis-de-um-objeto-javascript-c82442ac8cd5;
- https://stackoverflow.com/questions/684672/how-do-i-loop-through-or-enumerate-a-javascript-object;
- https://www.dofactory.com/javascript/advanced-objects;