OBJETOS

Object

Object

Todo objeto é criado com o construtor Object e por isso herda as propriedades e métodos do seu prototype.

```
const carro = {
  marca: 'Ford',
  ano: 2018,
}

const pessoa = new Object({
  nome: 'André',
  idade: 28,
})
```

Métodos de Object

`Object.create(obj, defineProperties)` retorna um novo objeto que terá como protótipo o objeto do primeiro argumento.

```
const carro = {
  rodas: 4,
  init(marca) {
    this.marca = marca;
    return this;
  },
  acelerar() {
    return `${this.marca} acelerou as ${this.rodas} rodas`;
  },
  buzinar() {
    return this.marca + ' buzinou';
  }
}

const honda = Object.create(carro);
honda.init('Honda').acelerar();
```

Object.assign()

`Object.assign(alvo, obj1, obj2)` adiciona ao alvo as propriedades e métodos enumeráveis dos demais objetos. O assign irá modificar o objeto alvo.

```
const funcaoAutomovel = {
  acelerar() {
    return 'acelerou';
  },
  buzinar() {
    return 'buzinou';
  },
}

const moto = {
  rodas: 2,
  capacete: true,
}

const carro = {
  rodas: 4,
  mala: true,
```

```
Object.assign(moto, funcaoAutomovel);
Object.assign(carro, funcaoAutomovel);
```

Object.defineProperties()

`Object.defineProperties(alvo, propriedades)` adiciona ao alvo novas propriedades. A diferença aqui é a possibilidade de serem definidas as características dessas propriedades.

```
const moto = {}
Object.defineProperties(moto, {
  rodas: {
    value: 2,
    configurable: false, // impede deletar e mudança de valor
    enumarable: true, // torna enumerável
  },
  capacete: {
    value: true,
    configurable: true,
    writable: false, // impede mudança de valor
  },
})

moto.rodas = 4;
delete moto.capacete;
moto;
```

> *Existe também o Object.defineProperty, para uma propriedade única.*

get e set

É possível definirmos diferentes comportamentos para get e set de uma propriedade. Lembrando que ao acionarmos uma propriedade `obj.propriedade`, a função get é ativada e ao setarmos `ob.propriedade = 'Valor'` a função de set é ativada.

```
const moto = {}
Object.defineProperties(moto, {
  velocidade: {
    get() {
      return this._velocidade;
    },
    set(valor) {
      this._velocidade = 'Velocidade ' + valor;
    }
  },
})

moto.velocidade = 200;
moto.velocidade;
// Velocidade 200
```

Object.getOwnPropertyDescriptors(obj)

Lista todos os métodos e propriedades de um objeto, com as suas devidas configurações.

```
Object.getOwnPropertyDescriptors(Array);
// Lista com métodos e propriedades e Array

Object.getOwnPropertyDescriptors(Array.prototype);
// Lista com métodos e propriedades do protótipo de Array

Object.getOwnPropertyDescriptor(window, 'innerHeight');
// Puxa de uma única propriedade
```

Object.keys(obj), Object.values(obj) Object.entries(obj)

`Object.keys(obj)` retorna uma array com as chaves de todas as propriedades diretas e enumeráveis do objeto. `Object.values(obj)` retorna uma array com os valores do objeto. `Object.entries(obj)` retorna uma array com array's contendo a chave e o valor.

```
Object.keys(Array);
// [] vazia, pois não possui propriedades enumeráveis

const carro = {
  marca: 'Ford',
  ano: 2018,
}
Object.keys(carro);
// ['marca', 'ano']
Object.values(carro);
// ['Ford', 2018]
Object.entries(carro);
// [['marca', 'Ford'], ['ano', 2018]]
```

Object.getOwnPropertyNames(obj)

Retorna uma array com todas as propriedades diretas do objeto (não retorna as do protótipo).

```
Object.getOwnPropertyNames(Array);
// ['length', 'name', 'prototype', 'isArray', 'from', 'of']

Object.getOwnPropertyNames(Array.prototype);
// [..., 'filter', 'map', 'every', 'some', 'reduce', ...]

const carro = {
  marca: 'Ford',
  ano: 2018,
}
Object.getOwnPropertyNames(carro);
// ['marca', 'ano']
```

Object.getPrototypeOf() e Object.is()

`Object.getPrototypeOf()`, retorna o protótipo do objeto. Object.is(obj1, obj2) verifica se os objetos são iguais e retorna true ou false.

```
const frutas = ['Banana', 'Pêra']
Object.getPrototypeOf(frutas);
Object.getPrototypeOf(''); // String

const frutas1 = ['Banana', 'Pêra'];
const frutas2 = ['Banana', 'Pêra'];

Object.is(frutas1, frutas2); // false
```

Object.freeze(), Object.seal(), Object.preventExtensions()

`Object.freeze()` impede qualquer mudança nas propriedades.
`Object.seal()` previne a adição de novas propriedades e impede que as atuais sejam deletadas.
`Object.preventExtensions()` previne a adição de novas propriedades.

```
const carro = {
  marca: 'Ford',
  ano: 2018,
}
Object.freeze(carro);
Object.seal(carro);
Object.preventExtensions(carro);

Object.isFrozen(carro); // true
Object.isSealed(carro); // true
Object.isExtensible(carro); // true
```

Propriedades e Métodos do Protótipo

Já que tudo em JavaScript é objeto, as propriedades abaixo estão disponíveis em todos os objetos criados a partir de funções construtoras. `{}.constructor` retorna a função construtora do objeto.

```
const frutas = ['Banana', 'Uva'];
frutas.constructor; // Array

const frase = 'Isso é uma String';
frase.constructor; // String
```

{}.hasOwnProperty('prop') e {}.propertyIsEnumerable('prop')

Verifica se possui a propriedade e retorna true. A propriedade deve ser direta do objeto e não do protótipo. O `{}.propertyIsEnumerable()` verifica se a propriedade é enumerável.

```
const frutas = ['Banana', 'Uva'];

frutas.hasOwnProperty('map'); // false
Array.hasOwnProperty('map'); // false
Array.prototype.hasOwnProperty('map'); // true

Array.prototype.propertyIsEnumerable('map'); // false
window.propertyIsEnumerable('innerHeight'); // true
```

{}.isPrototypeOf(valor)

Verifica se é o protótipo do valor passado.

```
const frutas = ['Banana', 'Uva'];

Array.prototype.isPrototypeOf(frutas); // true
```

{}.toString()

Retorna o tipo do objeto. O problema é toString() ser uma função dos protótipos de Array, String e mais. Por isso é comum utilizarmos a função direto do `Object.prototype.toString.call(valor)`.

```
const frutas = ['Banana', 'Uva'];
frutas.toString(); // 'Banana,Uva'
typeof frutas; // object
Object.prototype.toString.call(frutas); // [object Array]

const frase = 'Uma String';
frase.toString(); // 'Uma String'
typeof frase; // string
Object.prototype.toString.call(frase); // [object String]

const carro = {marca: 'Ford'};
carro.toString(); // [object Object]
typeof carro; // object
Object.prototype.toString.call(carro); // [object Object]

const li = document.querySelectorAll('li');
```

```
Object.prototype.toString.call(li); // [object NodeList]
```

Exercícios

```
// Crie uma função que verifique
// corretamente o tipo de dado

// Crie um objeto quadrado com
// a propriedade lados e torne
// ela imutável

// Previna qualquer mudança
// no objeto abaixo
const configuracao = {
  width: 800,
  height: 600,
  background: '#333'
}

// Liste o nome de todas
// as propriedades do
// protótipo de String e Array
```