

# CLASSES

---

Classes

## Constructor Function

---

Funções responsáveis pela criação de objetos. O conceito de uma função construtora de objetos é implementado em outras linguagens como Classes.

```
function Button(text, background) {  
    this.text = text;  
    this.background = background;  
}  
  
Button.prototype.element = function() {  
    const buttonElement = document.createElement('button');  
    buttonElement.innerText = this.text;  
    buttonElement.style.background = this.background;  
    return buttonElement;  
}  
  
const blueButton = new Button('Comprar', 'blue');
```

## Class

O ES6 trouxe uma nova sintaxe para implementarmos funções construtoras. Agora podemos utilizar a palavra chave `class`. É considerada `syntactical sugar`, pois por baixo dos panos continua utilizando o sistema de protótipos de uma função construtora para criar a classe.

```
class Button {  
    constructor(text, background) {  
        this.text = text;  
        this.background = background;  
    }  
    element() {  
        const buttonElement = document.createElement('button');  
        buttonElement.innerText = this.text;  
        buttonElement.style.background = this.background;  
        return buttonElement;  
    }  
}  
  
const blueButton = new Button('Comprar', 'blue');
```

## Class vs Constructor Function

---

```
class Button {  
    constructor(propriedade) {  
        this.propriedade = propriedade;  
    }  
    metodo1() {}  
    metodo2() {}  
}  
  
function Button(propriedade) {  
    this.propriedade = propriedade;  
}  
Button.prototype.metodo1 = function() {}  
Button.prototype.metodo2 = function() {}
```

# Constructor

O método `constructor(args) {}` é um método especial de uma classe. Nele você irá definir todas as propriedades do objeto que será criado. Os argumentos passados em `new`, vão direto para o constructor.

```
class Button {  
    constructor(text, background, color) {  
        this.text = text;  
        this.background = background;  
        this.color = color;  
    }  
}  
  
const blueButton = new Button('Clique', 'blue', 'white');  
// Button {text: 'Clique', background: 'blue', color: 'white'}
```

## Constructor Return

---

Por padrão a classe retorna `this`. Ou seja, `this` é o objeto criado com o `new Class`. Podemos modificar isso alterando o `return` do constructor, o problema é que perderá toda a referência do objeto.

```
class Button {  
    constructor(text) {  
        this.text = text;  
        return this.element(); // não fazer  
    }  
    element() {  
        document.createElement('button').innerText = this.text;  
    }  
}  
  
const btn = new Button('Clique');  
// <button>Clique</button>
```

## This

---

Assim como em uma função construtora, this faz referência ao objeto criado com new. Você pode acessar as propriedades e métodos da classe utilizando o this.

```
class Button {  
    constructor(text) {  
        this.text = text;  
    }  
    element() {  
        const buttonElement = document.createElement('button')  
        buttonElement.innerText = this.text;  
        return buttonElement;  
    }  
    appendElementTo(target) {  
        const targetElement = document.querySelector(target);  
        targetElement.appendChild(this.element());  
    }  
}  
  
const blueButton = new Button('Clique');  
blueButton.appendElementTo('body');
```

## Propriedades

---

Podemos passar qualquer valor dentro de uma propriedade.

```
class Button {  
  constructor(options) {  
    this.options = options;  
  }  
}  
  
const blueOptions = {  
  backgroundColor: 'blue',  
  color: 'white',  
  text: 'Clique',  
  borderRadius: '4px',  
}  
  
const blueButton = new Button(blueOptions);  
blueButton.options;
```

# Static vs Prototype

Por padrão todos os métodos criados dentro da classe irão para o protótipo da mesma. Porém podemos criar métodos diretamente na classe utilizando a palavra chave `static`. Assim como

`[ ].map()` é um método de uma array e `Array.from()` é um método do construtor Array.

```
class Button {  
    constructor(text) {  
        this.text = text;  
    }  
    static create(background) {  
        const elementButton = document.createElement('button');  
        elementButton.style.background = background;  
        elementButton.innerText = 'Clique';  
        return elementButton;  
    }  
}  
  
const blueButton = Button.create('blue');
```

## Static

---

Você pode utilizar um método `static` para retornar a própria classe com propriedades já pré definidas.

```
class Button {  
    constructor(text, background) {  
        this.text = text;  
        this.background = background;  
    }  
    element() {  
        const elementButton = document.createElement('button');  
        elementButton.innerText = this.text;  
        elementButton.style.background = this.background;  
        return elementButton  
    }  
    static createBlue(text) {  
        return new Button(text, 'blue');  
    }  
}  
  
const blueButton = Button.createBlue('Comprar');
```

# CLASSES

---

Get e Set

## Get e Set

---

Podemos definir comportamentos diferentes de get e set para um método.

```
const button = {  
    get element() {  
        return this._element;  
    }  
    set element(tipo) {  
        this._element = document.createElement(tipo);  
    }  
}  
  
button.element = 'button'; // set  
button.element; // get (<button></button>);
```

## Valor estático

---

Se definirmos apenas o get de um método, teremos então um valor estático que não será possível mudarmos.

```
const matematica = {  
  get PI() {  
    return 3.14;  
  }  
  
  matematica.PI; // get (3.14)  
  matematica.PI = 20; // nada acontece
```

## Set

---

Eu posso ter um método com set apenas, que modifique outras propriedades do meu objeto.

```
const frutas = {  
    lista: [],  
    set nova(fruta) {  
        this.lista.push(fruta);  
    }  
  
    frutas.nova = 'Banana';  
    frutas.nova = 'Morango';  
    frutas.lista; // ['Banana', Morango];
```

# Class

---

Assim como em um objeto, as classes podem ter métodos de get e set também.

```
class Button {  
    constructor(text, color) {  
        this.text = text;  
        this.color = color;  
    }  
    get element() {  
        const buttonElement = document.createElement('button');  
        buttonElement.innerText = this.text;  
        buttonElement.style.color = this.color;  
        return buttonElement;  
    }  
}  
  
const blueButton = new Button('Comprar', 'blue');  
blueButton.element; // retorna o elemento
```

# Class

---

Assim como em um objeto, as classes podem ter métodos de get e set também.

```
class Button {  
    constructor(text, color) {  
        this.text = text;  
        this.color = color;  
    }  
    get element() {  
        const buttonElement = document.createElement('button');  
        buttonElement.innerText = this.text;  
        buttonElement.style.color = this.color;  
        return buttonElement;  
    }  
}  
  
const blueButton = new Button('Comprar', 'blue');  
blueButton.element; // retorna o elemento
```

## Set e Class

---

Com o set podemos modificar apenas parte do elemento de get. É comum definirmos variáveis **privadas**, utilizando o underscore **\_privada**.

```
class Button {  
    constructor(text) {  
        this.text = text;  
    }  
    get element() {  
        const elementType = this._elementType || 'button';  
        const buttonElement = document.createElement(elementType);  
        buttonElement.innerText = this.text;  
        return buttonElement;  
    }  
    set element(type) {  
        this._elementType = type;  
    }  
}  
  
const blueButton = new Button('Comprar');  
blueButton.element; // retorna o elemento
```

# CLASSES

---

Extends

## Subclasses

---

É possível criarmos uma subclasse, esta irá ter acesso aos métodos da classe à qual ela estendeu através do seu protótipo.

```
class Veiculo {  
    constructor(rodas) {  
        this.rodas = rodas;  
    }  
    acelerar() {  
        console.log('Acelerou');  
    }  
}  
  
class Moto extends Veiculo {  
    empinar() {  
        console.log('Empinou com ' + this.rodas + ' rodas');  
    }  
}  
  
const honda = new Moto(2);  
honda.empinar();
```

# Métodos

Podemos escrever por cima de um método herdado.

```
class Veiculo {  
    constructor(rodas) {  
        this.rodas = rodas;  
    }  
    acelerar() {  
        console.log('Acelerou');  
    }  
}  
  
class Moto extends Veiculo {  
    acelerar() {  
        console.log('Acelerou muito');  
    }  
}  
  
const honda = new Moto(2);  
honda.acelerar();
```

# Super

É possível utilizar a palavra chave `super` para falarmos com a classe que pai e acessarmos os seus métodos e propriedades.

```
class Veiculo {  
    constructor(rodas) {  
        this.rodas = rodas;  
    }  
    acelerar() {  
        console.log('Acelerou');  
    }  
}  
  
class Moto extends Veiculo {  
    acelerar() {  
        super.acelerar();  
        console.log('Muito');  
    }  
}  
  
const honda = new Moto(2);  
honda.acelerar();
```

## Super e Constructor

Podemos utilizar o super para estendermos o método constructor.

```
class Veiculo {  
    constructor(rodas, combustivel) {  
        this.rodas = rodas;  
        this.combustivel = combustivel;  
    }  
}  
  
class Moto extends Veiculo {  
    constructor(rodas, combustivel, capacete) {  
        super(rodas, combustivel);  
        this.capacete = capacete;  
    }  
}  
  
const honda = new Moto(4, 'Gasolina', true);
```