ORIGAMID

# OBJETOS

Constructor Functions

# Objetos

Criar um objeto é simples, basta definirmos uma variável e iniciar a definição do seu valor com chaves `{}`. Mas e se precisarmos criar um novo objeto, com as mesmas características do anterior? É possível com o `Object.create`, mas veremos ele mais tarde.

```
const carro = {
  marca: 'Marca',
  preco: 0,
}

const honda = carro;
honda.marca = 'Honda';
honda.preco = 4000;

const fiat = carro;
fiat.marca = 'Fiat';
fiat.preco = 3000;
```

> *carro, fiat e honda apontam para o mesmo objeto.*

# Constructor Functions

Para isso existem as Constructor Functions, ou seja, Funções Construtoras que são responsáveis por construir novos objetos sempre que chamamos a mesma.

```
function Carro() {
  this.marca = 'Marca';
  this.preco = 0;
}

const honda = new Carro();
honda.marca = 'Honda';
honda.preco = 4000;
const fiat = new Carro();
fiat.marca = 'Fiat';
fiat.preco = 3000;
```

> *Usar Pascal Case, ou seja, começar com letra maiúscula.*

# new Keyword

A palavra chave `new` é responsável por criar um novo objeto baseado na função que passarmos a frente dela.

```
const honda = new Carro();

// 1 Cria um novo objeto
honda = {};

// 2 Define o protótipo
honda.prototype = Carro.prototype;

// 3 Aponta a variável this para o objeto
this = honda;

// 4 Executa a função, substituindo this pelo objeto
honda.marca = 'Marca';
honda.preco = 0;

// 5 Retorna o novo objeto
return honda = {
  marca: 'Marca',
```

```
}
```

## Parâmetros e Argumentos

Podemos passar argumentos que serão utilizados no momento da criação do objeto.

```
function Carro(marca, preco) {
  this.marca = marca;
  this.preco = preco;
}

const honda = new Carro('Honda', 4000);
const fiat = new Carro('Fiat', 3000);
```

## this Keyword

O `this` faz referência ao próprio objeto construído com a Constructor Function.

```
function Carro(marca, precoInicial) {
  const taxa = 1.2;
  const precoFinal = precoInicial * taxa;
  this.marca = marca;
  this.preco = precoFinal;
  console.log(this);
}

const honda = new Carro('Honda', 2000);
```

> *Variáveis dentro da Constructor estão "protegidas".*

## Exemplo Real

Quando mudamos a propriedade seletor, o objeto Dom irá passar a selecionar o novo seletor em seus métodos.

```
const Dom = {
  seletor: 'li',
  element() {
    return document.querySelector(this.seletor);
  },
  ativo() {
    this.element().classList.add('ativo');
  },
}

Dom.ativo(); // adiciona ativo ao li
Dom.seletor = 'ul';
Dom.ativo(); // adiciona ativo ao ul
```

# Constructor Function Real

Um objeto criado com uma Constructor, não irá influenciar em outro objeto criado com a mesma Constructor.

```
function Dom() {
  this.seletor = 'li';
  const element = document.querySelector(this.seletor);
  this.ativo = function() {
    element.classList.add('ativo');
  };
}

const lista = new Dom();
lista.seletor = 'ul';
lista.ativo();

const lastLi = new Dom();
lastLi.seletor = 'li:last-child';
lastLi.ativo();
```

# Lembre-se de usar parâmetros

```
function Dom(seletor) {
  const element = document.querySelector(seletor);
  this.ativo = function(classe) {
    element.classList.add(classe);
  };
}

const lista = new Dom('ul');
lista.ativo('ativo');

const lastLi = new Dom('li:last-child');
lastLi.ativo('ativo');
```

# Exercícios

```
// Transforme o objeto abaixo em uma Constructor Function
const pessoa = {
  nome: 'Nome pessoa',
  idade: 0,
  andar() {
    console.log(this.nome + ' andou');
  }
}

// Crie 3 pessoas, João - 20 anos,
// Maria - 25 anos, Bruno - 15 anos


// Crie uma Constructor Function (Dom) para manipulação
// de listas de elementos do dom. Deve conter as seguintes
// propriedades e métodos:
//
// elements, retorna NodeList com os elementos selecionados
// addClass(classe), adiciona a classe a todos os elementos
// removeClass(classe), remove a classe a todos os elementos
```