Constructor Function 1

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,
}
```

Abriu as chaves eu defino as propriedades e métodos dele.

Propriedade -> :
Método -> ()

Tudo é objeto, tudo o que a gente acessa na manipulação do DOM era sempre um Objeto, sempre colocando . propriedade ou método:

```
const d = new Date("2021-03-25");
d.getFullYear();
```

No exemplo acima eu criei um objeto chamado carro com as propriedades marca e preco e aí nesse seguinte exemplo:

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

Meu objetivo é criar um novo objeto baseado no anterior. A varíavel honda aponta pra carro e aí eu vou alterar a propriedade marca e a propriedade preco.

Faço o mesmo no segundo exemplo:

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

Não vai gerar um erro, mas não vai acontecer o que eu quero.

Output:

```
carro
▶ {marca: 'Marca', preco: 0}
```

Agora, imagina que eu quero usar vários carros, vários objetos carros que possuem marca e preco também então eu vou ter que sempre fazer a mesma coisa:

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

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

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

Eu não quero isso porque isso está se repetindo estou criando as mesmas propriedades e se tiver métodos depois eu vou ter que repetir. Então eu quero uma forma de copiar o objeto carro e somente substituir os valores das propriedades. Você poderia pensar assim:

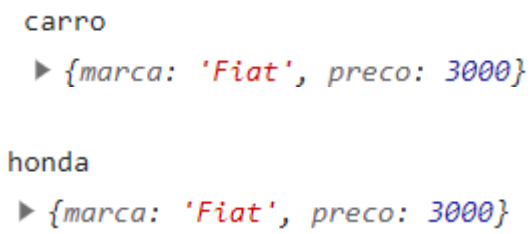
```
const honda = carro;
honda.marca = 'Honda';
honda.preco = 5000;
```

```
honda
▶ {marca: 'Honda', preco: 5000}
```

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

```
fiat
▶ {marca: 'Fiat', preco: 3000}
```

Aparentemente está funcionando, mas o que está acontecendo é que quando eu estou atribuindo honda a carro eu estou falando que isso aponta para o objeto carro, ou seja, honda é exatamente o que está em carro se eu for verificar carro:



Porque eu modifiquei na última linha.

Então na verdade tudo que eu estou fazendo é alterando o objeto carro e não copiando ele.

Para fazer essa cópia, esse clone vamos precisar de uma Função Construtora.

CONSTRUCTOR FUNCTIONS

Contruísmos ela assim:

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

Ela é uma função normal só que podemos fazer referência a ela mesma utilizando o **this** e para executar essa função temos que usar a palavra **new**:

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

Eu vou criar um **novo** carro.

Um padrão é que sempre que for uma Função Construtora você utiliza o Pascal case você nomeia com letra maiúscula e depois você continua com o Camel case. Isso é um só padrão pra identificar que é uma Função Construtora, mas se for minúsculo não tem problema é só o nome de uma função.

Leitura:
https://www.alura.com.br/artigos/convencoes-nomenclatura-camel-pascal-kebab-snake-case

```
function Car() {

}

const honda = Car();
```

Assim eu estou colocando em honda o que retorna da função Car(). Uma função vazia sem a palavra *return* sempre retorna *undefined*.

```
Car()
undefined

honda
undefined
```

```
function Car() {
  return 'Hello';
}

const honda = Car();
```

```
honda
'Hello'
```

Com o **new** o honda agora é um objeto:

```
const honda = new Car();
```

```
honda
▶ Car {}
```

Esse objeto é do tipo *Car,* então vamos chamar ele de Construtor de honda. Então honda é um objeto que vem do Construtor *Car*, dito isso agora eu posso criar uma propriedade:

```
function Car() {
  return 'Hello';
```

```
}

const honda = new Car();
honda.color = 'Red';
```

```
honda
▼ Car {color: 'Red'} i
    color: "Red"
  ▶ [[Prototype]]: Object
```

Também posso criar um método:

```
honda.andar = function() {
  console.log('Andou');
}
```

```
honda.andar()
Andou
undefined
```

Lembrando que o método eu apenas atribuo uma função e é retornado *undefined* porque eu não tenho nenhum return na minha função;

Ao invés de eu criar as propriedades e métodos assim eu vou criar dentro da função construtora Car() pra que quando eu criar o objeto honda ele já venha com tudo.

```
function Car() {
  console.log(this);
}
```

```
▼ Car {} i
  ▶ [[Prototype]]: Object
```

this é simplesmente o objeto carro;

Então eu posso criar as propriedades e métodos usando o **this** que faz referência ao construtor Car():

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

```
honda
▼ Car {marca: 'Marca', preco: 0} i
    marca: "Marca"
    preco: 0
  ▶ [[Prototype]]: Object
```

Se eu criar um novo carro assim:

```
const fiat = new Car();
```

```
fiat
▶ Car {marca: 'Marca', preco: 2000}
```

Vai se manter igual porque eu não atribui nada de diferente.

Mas se eu alterar a propriedade marca o resultado se altera:

```
fiat.marca = 'Fiat';
```

```
fiat
▶ Car {marca: 'Fiat', preco: 2000}
```

E a variável honda não se altera:

```
honda
▼ Car {marca: 'Marca', preco: 0} i
    marca: "Marca"
    preco: 0
  ▶ [[Prototype]]: Object
```

Agora sim eu tenho objetos totalmente diferentes porque o **new** permite que eu construa um novo objeto para o honda e para o fiat e em quantos carros eu quiser baseado na Função Construtora Car().

NEW KEYWORD

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

Ele possui 5 etapas ele vai ser sempre responsável por criar esse novo objeto da função.

1°: Ele cria um novo objeto vazio que vai ser pelo nome da variável que você colocou antes pra definir o objeto, neste caso, honda.

```
const honda = new Car();

honda = {};
```

2°: Ele vai definir o protótipo de honda como protótipo de Car

```
const honda = new Car();

honda.prototype = Carro.prototype;
```

Veremos mais sobre protótipos adiante, mas basicamente ele vai herdar os métodos e propriedades da Função Construtora etc.

3°: Ele vai apontar a variável **this** para o objeto, para o carro honda como se estivesse fazendo this = honda;

```
this = honda;
```

4°: Executa a função, substituindo this pelo objeto

```
honda.marca = 'Marca';
honda.preco = 0;
```

Ele vai fazer tudo isso só pelo fato de ter colocado o **new** na frente;

5°: Retorna o novo objeto

```
return honda = {
  marca: 'Marca',
  preco: 0,
}
```

O objeto que estava vazio agora tem os valores atribuídos;

Então, é uma função ainda que está retornando um valor só que esse valor que está sendo retornado dela é um objeto com propriedades e métodos que você pode definir dentro dela.

PARÂMETROS E ARGUMENTOS

E ao invés de criarmos assim:

```
function Car() {
  this.marca = 'Marca';
  this.preco = 2000;
}

const honda = new Car();
const fiat = new Car();
fiat.marca = 'Fiat';
```

Eu posso definir parâmetros na função Car() pra depois passar os argumentos dentro:

```
function Car(marcaAtribuida, precoAtribuido) {
  this.marca = marcaAtribuida;
  this.preco = precoAtribuido;
}

const honda = new Car('Honda', 5000);
const fiat = new Car('Fiat', 3000);
```