THIS

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

```
function Car2(marca, precoInicial) {
  const taxa = 1.2;
  const precoFinal = precoInicial * taxa;
  this.marca = marca;
  this.preco = precoFinal;
  console.log(this);
}

const honda = new Car2('Honda', 2000);
```

```
▼ Car2 {marca: 'Honda', preco: 2400} i
    marca: "Honda"
    preco: 2400
  ▶ [[Prototype]]: Object
```

Importante: o preçoFinal vai ser o preço do carro com a taxa e o objeto honda só tem acesso a variável a marca e ao preço, porque foi o que eu escolhi exportar nele no objeto. Enquanto que a taxa e o precoFinal ele não vai ter acesso somente se eu colocar o *this* que aí sim ele vai passar como propriedade.

```
this.taxa = 1.2;
const precoFinal = precoInicial * 1.2;
```

```
▼ Car2 {taxa: 1.2, marca: 'Honda', preco: 2400} i
    marca: "Honda"
    preco: 2400
    taxa: 1.2
  ▶ [[Prototype]]: Object
```

Exemplo Real

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

Quando você vai criar um objeto na vida real dificilmente você vai realizar esse objeto tipo pessoa se você for puxar de um banco de dados vai vim um banco de dados com pessoas, ou seja, você não vai criar um objeto a parte disso assim como não vai ter um objeto carro e por aí vai.

Um Objeto real que a gente pode criar é um do jeito que faça a gente manipular o nosso DOM:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Constructor Functions</title>
</head>
<body>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>
  <script src="script.js"></script>
</body>
</html>
```

```
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
```

Ainda não é um objeto construtor, é um objeto simples;

O meu objeto tem a propriedade seletor junto com o método element() que vai simplesmente retornar esse objeto do DOM baseado na propriedade com base nesse seletor e eu tenho o método ativo() que vai simplesmente adicionar a classe ativo a esse meu elemento.

```
element() {
    return document.querySelector(this.seletor);
}
```

É a mesma coisa que:

```
element: function() {

}
```

É só um atalho;

Lembrando que método é uma função!

Quando eu estou dentro de um objeto e eu quero fazer referência a alma propriedade do objeto eu tenho que chamar a palavra *this*:

```
element() {
    console.log(seletor);
}
```

```
> DOM
< ▶ {seletor: 'li', element: ƒ}
> DOM.element()
✖ ▶ Uncaught ReferenceError: seletor is not defined
      at Object.element (script.js:24:17)
      at <anonymous>:1:5
```

```
element() {
    console.log(this.seletor);
}
```

```
> DOM.element()
  li
< undefined
```

Para selecionar o elemento:

```
const DOM = {
  seletor: 'li',
  element() {
```

```
    return document.querySelector(this.seletor);
  }
}
```

```
> DOM.element()
<li>
    ::marker
    "Item 1"
</li>
```

Lembrando que esse elemento eu já posso usar todas as propriedades e métodos desse elemento:

```
> DOM.element().classList.add('teste');
undefined
```

```
<ul>
  <li class="teste"> == $0
    ::marker
    "Item 1"
  </li>
```

O método ativo() você poderia fazer assim:

```
ativo() {
    const elementoSelecionado = document.querySelector(this.seletor);
    elementoSelecionado.classList.add('ativo');
},
```

```
> DOM.ativo()
undefined
```

Dá *undefined* porque a função não está retornando nada. Mas se a gente for no HTML, lá está a classe:

```
<ul>
  <li class="ativo">
    ::marker
    "Item 1"
  </li>
```

Só que eu já tenho um método que retorna o element, então eu não preciso selecionar tudo de novo o método que retorna o element eu posso acessar ele assim:

```
ativo() {
    const elementoSelecionado = this.element();
    elementoSelecionado.classList.add('ativo');
}
```

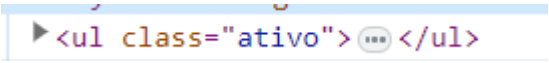
E o resultado é o mesmo porque o método element() simplesmente retorna o elemento e eu poderia passar direto assim:

```
ativo() {
    this.element().classList.add('ativo');
}
```

O resultado continua o mesmo;

Agora imagine que eu quero ativar o meu ul, então eu poderia fazer:

```
DOM.seletor = 'ul';
DOM.ativo();
```



Se eu quisesse adicionar na li:

```
DOM.ativo();
```

Ele ativa na li também e como eu mudei o seletor logo depois ele vai adicionar na ul.

```
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
```

Esse é o problema da gente criar um objeto aqui porque eu quero reaproveitá-lo Porque eu quero ativar a ul, depois a li etc e eu vou ter que ficar toda hora mudando de seletor e totalmente o meu objeto vai mudar totalmente sempre vai fazer referência a coisa nova:

```
> DOM
< ▶ {seletor: 'ul', element: ƒ, ativo: ƒ}
```

Agora o objeto está com um *ul* dentro dele e não é isso o que eu quero; para isso existem as funções construtoras.

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

DOM.ativo();
DOM.seletor = 'ul';
DOM.ativo();
```

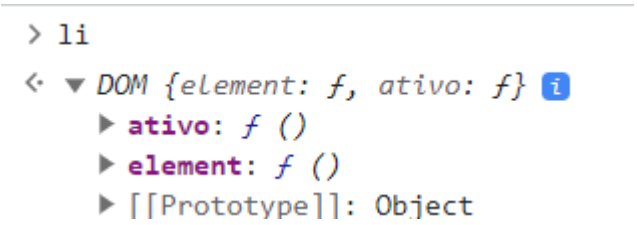
Para transformar esse código em uma função construtora:

```
function DOM(seletor) {
  this.element = function() {
    return document.querySelector(seletor);
  }
  this.ativo = function() {
    this.element().classList.add('ativo');
  }
}
```

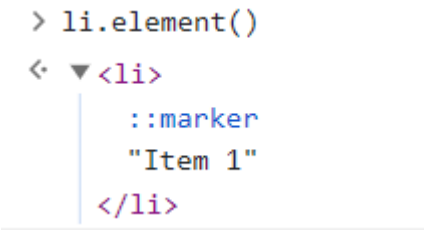
Para eu usar a função construtora agora:

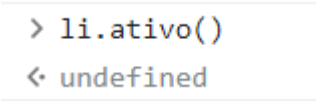
```
const li = new DOM('li');
```

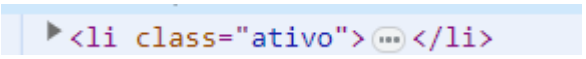
O *li* vai ser selecionado no seletor, depois vai se transformar em *this.element* e vai adicionar a classe ‘ativar’ se eu ativar a função ativar.



Agora a *li* é um objeto do construtor DOM e tem o método ativo() e element().







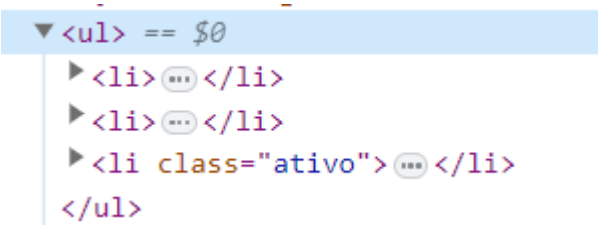
Agora eu posso selecionar o elemento ul:

```
const ul = new DOM('ul');
```

Agora eu tenho dois objetos *ul* e *li* ambos com os mesmos métodos e propriedades.

Então eu posso passar qualquer tipo de seletor:

```
const liLastChild = new DOM('li:last-child');
liLastChild.ativo();
```



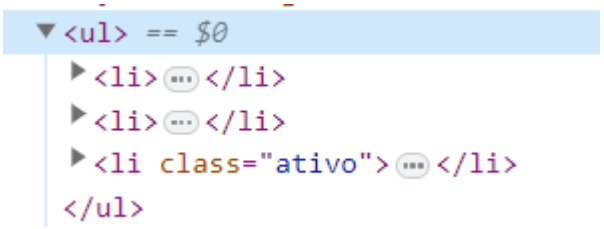
Então, um objeto criado com uma Constructor, não irá influenciar em outro objeto criado com a mesma Constructor.

Lembre-se de usar parâmetros

Você pode usar parâmetros dentro do método:

```
this.ativo = function(classe) {
    this.element().classList.add(classe);
}
```

```
const liLastChild = new DOM('li:last-child');
liLastChild.ativo('ativo');
```



Então você pode ter vários parâmetros e é assim que funciona como por exemplo todos aqueles métodos e propriedades do *document*.

A partir daí que está o poder do objeto porque a partir de alguns parâmetros novos você vai criar um objeto com essa nova seleção que você criou esses novos parâmetros.

Código completo:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Constructor Functions</title>
</head>
<body>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
```

```
    <li>Item 3</li>
  </ul>
  <script src="script-exercicio.js"></script>
</body>
</html>
```

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

const li = new DOM('li');
const ul = new DOM('ul');

const liLastChild = new DOM('li:last-child');
liLastChild.ativo('ativo');
```

Esse é o poder que a gente tem usando as funções construtoras Por exemplo alguma coisa que manipula o DOM, por exemplo, você poderia adicionar uma classe a todos os itens de uma lista muitas vezes a gente teve que fazer loops direto sempre tendo que fazer o loop, sempre que a gente fosse remover tudo. Com isso não iria precisar mais a gente só selecionaria o elemento do DOM a partir da palavra **new** e a partir daí a gente faria o que quisesse.