OBJETOS

Function

Function

Toda função é criada com o construtor Function e por isso herda as suas propriedades e métodos.

```
function areaQuadrado(lado) {
  return lado * lado;
}

const perimetroQuadrado = new Function('lado', 'return lado * 4');
```

Propriedades

`Function.length` retorna o total de argumentos da função.
`Function.name` retorna uma string com o nome da função.

```
function somar(n1, n2) {
  return n1 + n2;
}

somar.length; // 2
somar.name; // 'somar'
```

function.call()

`function.call(this, arg1, arg2, ...)` executa a função, sendo possível passarmos uma nova referência ao `this` da mesma.

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

function descricaoCarro() {
  console.log(this.marca + ' ' + this.ano);
}

descricaoCarro() // undefined undefined
descricaoCarro.call() // undefined undefined
descricaoCarro.call(carro) // Ford 2018
```

This

O valor de this faz referência ao objeto criado durante a construção do objeto (Constructor Function). Podemos trocar a referência do método ao this, utilizando o `call()`.

```
const carros = ['Ford', 'Fiat', 'VW'];

carros.forEach((item) => {
  console.log(item);
}); // Log de cada Carro

carros.forEach.call(carros, (item) => {
  console.log(item);
}); // Log de cada Carro

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

carros.forEach.call(frutas, (item) => {
  console.log(item);
}); // Log de cada Fruta
```

Exemplo Real

O objeto atribuído a `lista` será o substituído pelo primeiro argumento de `call()`

```
function Dom(seletor) {
  this.element = document.querySelector(seletor);
};

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

const lista = new Dom('ul');
lista.ativo('ativar');
console.log(lista);
```

O Objeto deve ser parecido

O novo valor de this deve ser semelhante a estrutura do valor do this original do método. Caso contrário o método não conseguirá interagir de forma correta com o novo this.

```
const novoSeletor = {
  element: document.querySelector('li')
}

Dom.prototype.ativo.call(novoSeletor, 'ativar');
```

Array's e Call

É comum utilizarmos o `call()` nas funções do protótipo do construtor Array. Assim podemos estender todos os métodos de Array à objetos que se parecem com uma Array (array-like).

```
Array.prototype.mostreThis = function() {
  console.log(this);
}

const frutas = ['Uva', 'Maçã', 'Banana'];
frutas.mostreThis(); // ['Uva', 'Maçã', 'Banana']

Array.prototype.pop.call(frutas); // Remove Banana
frutas.pop(); // Mesma coisa que a função acima
```

Array-like

HTMLCollection, NodeList e demais objetos do Dom, são parecidos com uma array. Por isso conseguimos utilizar os mesmos na substituição do this em call.

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

const filtro = Array.prototype.filter.call(li, function(item) {
  return item.classList.contains('ativo');
});

filtro; // Retorna os itens que possuem ativo
```

function.apply()

O `apply(this, [arg1, arg2, ...])` funciona como o call, a única diferença é que os argumentos da função são passados através de uma array.

```
const numeros = [3,4,6,1,34,44,32];
Math.max.apply(null, numeros);
Math.max.call(null, 3, 4, 5, 6, 7, 20);

// Podemos passar null para o valor
// de this, caso a função não utilize
// o objeto principal para funcionar
```

Apply vs Call

A única diferença é a array como segundo argumento.

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

function itemPossuiAtivo(item) {
  return item.classList.contains('ativo');
}

const filtro1 = Array.prototype.filter.call(li,
itemPossuiAtivo);
const filtro2 = Array.prototype.filter.apply(li,
[itemPossuiAtivo]);
```

function.bind()

Diferente de call e apply, `bind(this, arg1, arg2, ...)` não irá executar a função mas sim retornar a mesma com o novo contexto de this.

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

const filtrarLi = Array.prototype.filter.bind(li,
function(item) {
  return item.classList.contains('ativo');
});

filtrarLi();
```

Argumentos e Bind

Não precisamos passar todos os argumentos no momento do bind, podemos passar os mesmos na nova função no momento da execução da mesma.

```
const carro = {
  marca: 'Ford',
  ano: 2018,
  acelerar: function(aceleracao, tempo) {
    return `${this.marca} acelerou ${aceleracao} em ${tempo}`;
  }
}
carro.acelerar(100, 20);
// Ford acelerou 100 em 20

const honda = {
  marca: 'Honda',
};
const acelerarHonda = carro.acelerar.bind(honda);
acelerarHonda(200, 10);
// Honda acelerou 200 em 10
```

Argumentos Comuns

Podemos passar argumentos padrões para uma função e retornar uma nova função.

```
function imc(altura, peso) {
  return peso / (altura * altura);
}

const imc180 = imc.bind(null, 1.80);

imc(1.80, 70); // 21.6
imc180(70); // 21.6
```

Exercícios

```
<section>
  <p>Lobo-cinzento (nome científico:Canis lupus) é uma espécie de mamífero canídeo do gênero Canis. É um sobrevivente da Era do Gelo, originário do Pleistoceno Superior, cerca de 300 mil anos atrás. É o maior membro remanescente selvagem da família canidae.</p>
  <p>Os lobos-cinzentos são tipicamente predadores ápice nos ecossistemas que ocupam. Embora não sejam tão adaptáveis à presença humana como geralmente ocorre com as demais.</p>
  <p>O peso e tamanho dos lobos variam muito em todo o mundo, tendendo a aumentar proporcionalmente com a latitude.</p>
  <p>Os lobos são capazes de percorrer longas distâncias com uma velocidade média de 10 quilômetros por hora e são conhecidos por.</p>
</section>
```

```
// Retorne a soma total de caracteres dos
// parágrafos acima utilizando reduce
```

```
// html, com os seguintes parâmetros
// tag, classe e conteudo.


// Crie uma nova função utilizando a anterior como base
// essa nova função deverá sempre criar h1 com a
// classe titulo. Porém o parâmetro conteudo continuará
dinâmico
```