Eu posso fazer:

```
// new Object -> Object.prototype
const objA = {
    keyA: 'A'
    // __proto__: Object.prototype
};

const objB = {
    keyB: 'B'
    // __proto__: objA
};
```

Ao invés de __proto__: Object.prototype eu posso fazer -> __proto__: objA

Assim:

```
Object.setPrototypeOf(objB, objA);
console.log(objB.keyA);
```

Então mesmo eu não tendo a chave **a** em **b** eu posso acessá-la pelo objB;

```
// new Object -> Object.prototype
const objA = {
    keyA: 'A'
    // __proto__: Object.prototype
};

const objB = {
    keyB: 'B'
    // __proto__: objA
};

const objC = new Object();
objC.keyC = 'C';

Object.setPrototypeOf(objB, objA);
Object.setPrototypeOf(objC, objB);
console.log(objC.keyA);
```

Assim eu posso reaproveitar códigos que estão em outros objetos.

Não é recomendável usar a propriedade __proto__ !

Se eu quiser acessar o proto de algum objeto eu posso usar também a função:

Object.getPrototypeOf(objA);

Quando eu vi Getters e Setters aqui claramente tem um getter e aí o set é o que seta o prototype é ele quem configura o prototype.

Agora a gente vai criar métodos dentro do prototype da função!

```
function product(name, price) {
    this.name = name;
    this.price = price;
}

product.prototype.discount = function(percentual) {
    this.price = this.price - (this.price * (percentual / 100));
};

product.prototype.aumento = function(percentual) {
    this.price = this.price + (this.price * (percentual / 100));
};

const p1 = new product('Shirt', 90);
p1.discount(100);
p1.aumento(100);
console.log(p1);
```

Agora imagina que eu quero pegar um objeto produto e reaproveitar essas funções nesse objeto produto eu vou ter que usar o próprio prototype das funções pra setar o objeto literal p2 eu vou setar o prototype dele como sendo prototype das funções:

```
const p2 = {
    name: 'Cup',
    price: 15
};
Object.setPrototypeOf(p2, product.prototype);
p2.aumento(15);
console.log(p2);
```

O construtor do meu produto literal (p2) agora é a minha função produto;

Resultado:

```
product { name: 'Cup', price: 17.25 }
```

Tá sendo exibido exatamente como se fosse um produto criado da minha função construtora;

Uma outra maneira que eu posso fazer é já criar um objeto e já setar o prototype dele de cara:

```
const p3 = Object.create();
```

Essa função pode receber algumas coisas ela pode receber o objeto literal:

```
const p3 = Object.create(Object.prototype);
```

Então se eu crio um objeto e passo como prototype o Object.prototype eu tenho um objeto normal, vazio;

Resultado: {}

```
const p3 = Object.create(product.prototype);
console.log(p3);
```

Agora o construtor do p3 vai ser o produto, mas o problema de fazer isso nesse objeto específico é que eu não tenho as chaves ainda.

Resultado: product {}

Então crio a chave preço:

```
const p3 = Object.create(product.prototype);
p3.price = 110;
p3.aumento(5);
console.log(p3);
```

Outra coisa que eu posso fazer é já setar os atributos:

```
const p3 = Object.create(product.prototype, {
    size: {
        writable: true,
        configurable: true,
        enumerable: true,
        value: 42
    },
```

```
    color: {
        writable: true,
        configurable: true,
        enumerable: true,
        value: 'Red'
    }
});
```

Eu já estou criando as chaves no objeto p3 com as configurações e aí eu poderia também ao invés de fazer isso eu poderia chamar uma dessas chaves de price:

```
const p3 = Object.create(product.prototype, {
    price: {
        writable: true,
        configurable: true,
        enumerable: true,
        value: 90
    },
    color: {
        writable: true,
        configurable: true,
        enumerable: true,
        value: 'Red'
    }
});
```

Agora eu posso chamar o p3 porque agora ele tem a chave price:

```
p3.aumento(10);
console.log(p3);
```

Resultado:

```
product { price: 99, color: 'Red' }
```

Então eu estou criando o objeto, setando o prototype dele quem que vai ser o prototype do objeto neste caso p3 entre aspas quem vai ser o pai desse objeto que estou criando e depois de {} vai vir as chaves do objeto e as configurações dele assim como eu vi na aula de DefineProperty e DefineProperties.