Eles seriam uma maneira de proteger a minha propriedade, por exemplo, a propriedade stock só pode receber números se eu recebesse uma string eu poderia quebrar o meu programa.

O Getter é para obter o valor e o Setter para setar, configurar o valor;

Eu devo apagar a propriedade value e writable, o valor eu não vou precisar pois eu vou utilizar o Getter e o Setter para trabalhar no valor tanto para obter e configura-lo. E o writable não faz sentido se tenho um método que vai trabalhar no meu valor.

Configuro o Getter da seguinte forma:

```
          get: function ()
             return stock;
        }
```

Resultado:

```
product { name: 'Shirt', price: 10, stock: [Getter] }
```

O meu estoque não está sendo exibido ele é exibido como [Getter], ou seja, ele é um método que vai me retornar um valor. Para eu ver o valor de estoque (executar o Getter) eu faria:

```
console.log(p1.stock);
```

Então geralmente o trabalho do Getter vai ser só pegar o valor dele no momento e mostra-lo.

Já o Setter vai modificar o valor dentro e é nele que eu vou validar esse valor;

Mas devo tomar cuidado se eu for trabalhar com propriedades do objeto:

this.stock

Porque eu posso fazer o meu programar gerar um loop se eu mando **this.stock** dentro de set toda vez que eu chama-lo o JavaScript vai chamar essa função várias vezes e vai virar um loop. Então se eu quiser trabalhar com propriedades do objeto cria uma varíavel e trabalha nessa outra variável privada, o Getter e o Setter vão manipular a minha variável privada.

Como eu já tenho o parâmetro estoque então eu não preciso criar uma variável específica para isso;

Mas cabe o seguinte exemplo:

```
function product(name, price, stock) { // Função construtora
    this.name = name;
    this.price = price; // Estou atrelando a variável price no objeto

    let stockPrivate = stock;
    Object.defineProperty(this, 'stock', {
        enumerable: true, // mostra a chave
        configurable: true, // configurável
        get: function () {
            return stockPrivate;
        },
        set: function (value) {
            stockPrivate = value;
        }
    });
    /*
    Object.defineProperties(this, {
        name: {
        enumerable: true, // mostra a chave
        value: name, // valor da chave
        writable: false, // pode alterar
        configurable: true // configurável
        },
        price: {
        enumerable: true, // mostra a chave
        value: price, // valor da chave
        writable: false, // pode alterar
        configurable: true // configurável
        },
    });
    */
}
```

Fazendo isso eu já estou alterando o valor do meu estoque.

```
p1.stock = 'anything';
```

Eu só quero que essa mensagem aconteça se o valor que estou setando (set) for um número e não uma string e antes de setar o valor vou fazer a seguinte condição:

```
set: function (value) {
```

```
            if (typeof value !== 'number') {
                console.log('Hello');
                return;
            }
            stockPrivate = value;
        }
```

Eu protegi a minha variável stockPrivate, mas mais importante que a mensagem de Hello eu protegi a minha variável ela não será mais alterada enquanto eu não enviar um número.

```
p1.stock = 500;
```

Resultado: 500

Eu poderia lançar uma exceção:

```
if (typeof value !== 'number') {
                throw new TypeError('Hello'); // da erro porque eu não
disse qual o erro eu queria que fosse mostrado
            }
```

Na função Factory eu posso fazer direto no objeto:

```
function product(name, price, stock) { // Função construtora
    this.name = name;
    this.price = price; // Estou atrelando a variável price no objeto

    let stockPrivate = stock;
    Object.defineProperty(this, 'stock', {
        enumerable: true, // mostra a chave
        configurable: true, // configurável
        get: function () {
            return stockPrivate;
        },
        set: function (value) {
            if (typeof value !== 'number') {
                throw new TypeError('Hello');
            }
            stockPrivate = value;
        }
    });
    /*
```

```
    Object.defineProperties(this, {
        name: {
        enumerable: true, // mostra a chave
        value: name, // valor da chave
        writable: false, // pode alterar
        configurable: true // configurável
        },
        price: {
        enumerable: true, // mostra a chave
        value: price, // valor da chave
        writable: false, // pode alterar
        configurable: true // configurável
        },
    });
    */
}

function createProduct(name) {
    return {
        get name() {
            return name;
        },
        set name(value) {
            value = value.replace('thing', '');
            name = value;
        }
    }
}

//const p1 = new product('Shirt', 10, 3);
//console.log(p1);
//p1.stock = 500;
//console.log(p1.stock);

const p2 = createProduct('Shirt');
p2.name = 'anything';
console.log(p2.name);
```

Getters e Setters na função constrututora;