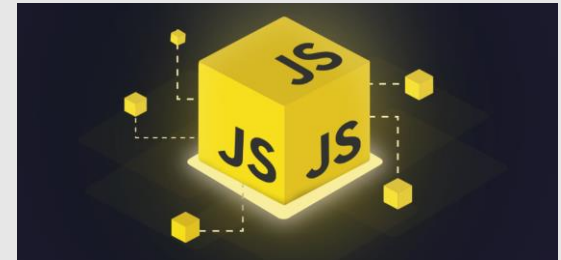


# Desenvolvimento Web

## Unidade 2 – Parte 4

### JS – Objetos e Coleções



Prof. Aparecido V. de Freitas  
Doutor em Engenharia  
da Computação pela EPUSP  
[aparecido.freitas@online.uscs.edu.br](mailto:aparecido.freitas@online.uscs.edu.br)  
[aparecidovfreitas@gmail.com](mailto:aparecidovfreitas@gmail.com)

# Objetos

- ❖ Na Linguagem JavaScript, um objeto é uma **coleção** não ordenada de pares **chave-valor**, onde as chaves podem ser **strings** ou **símbolos**, e os **valores** podem ser qualquer tipo de dado, incluindo outros objetos ou funções;
- ❖ **Objetos** são uma das principais estruturas de dados em **JavaScript** e são fundamentais para a linguagem, visto que quase tudo em **JavaScript** é, de alguma forma, um **objeto**.



# Objetos

- ❖ Pode-se pensar em objetos como uma **Tabela Hash**: nada mais que um **agrupamento de pares nome-valor**;
- ❖ **Valores** podem ser dados ou funções.



# Tipos Primitivos e Object

- ❖ **Object** é um dos tipos fundamentais em **JavaScript**;
- ❖ **JavaScript** tem um conjunto de **tipos primitivos** e **'object'** como um tipo complexo;
- ❖ Tipos primitivos em **JavaScript** incluem:

- `'string'`
- `'number'`
- `'boolean'`
- `'null'`
- `'undefined'`
- `'symbol'` (introduzido no ES6)
- `'bigint'` (introduzido no ES11/ES2020)

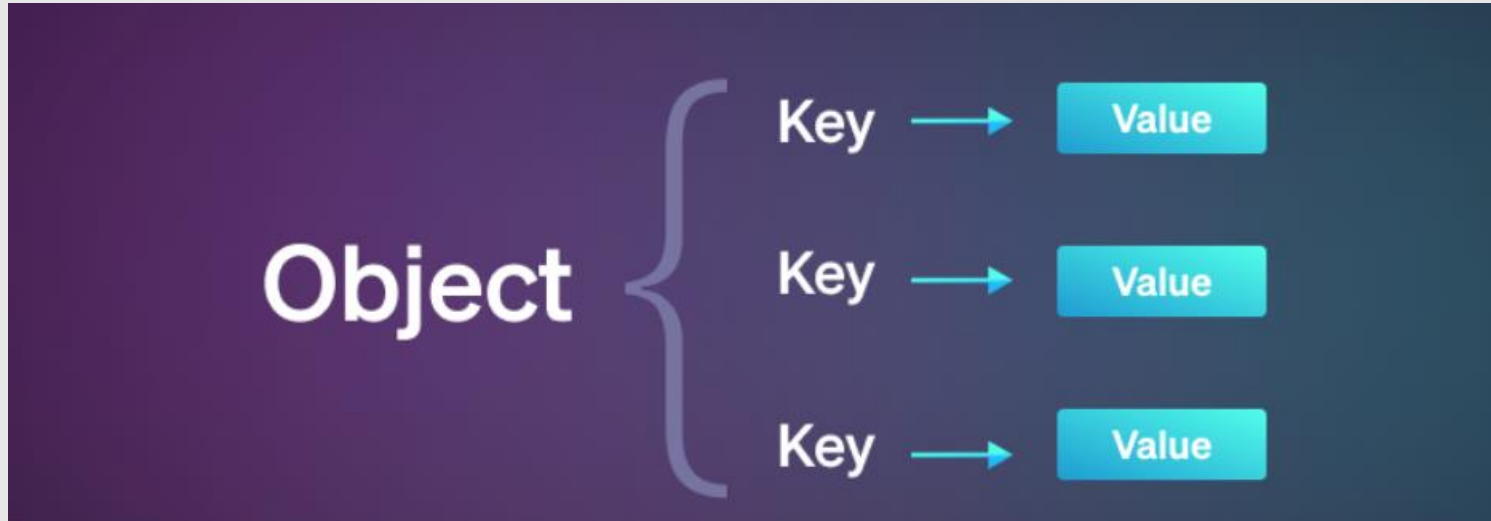
# Tipos Primitivos e Object

Cada um dos tipos primitivos, exceto ``null`` e ``undefined``, tem um objeto correspondente, também conhecido como objeto wrapper, ao qual pode ser convertido:

- ``String`` para ``string``
- ``Number`` for ``number``
- ``Boolean`` para ``boolean``
- ``Symbol`` para ``symbol``
- ``BigInt`` para ``bigint``

# Object

- ❖ **Object** é um dos tipos fundamentais em **JavaScript**;
- ❖ **Object** é um o tipo complexo em **JavaScript** e pode ser usado para representar uma **coleção** de dados e funcionalidades mais complexas;







# Como os objetos são criados?

# Criação de Objetos

## 1. Notação Literal:

javascript

```
let pessoa = {  
  nome: "Ana",  
  idade: 30  
};
```



# Criação de Objetos

## 2. Construtor Object:

javascript

```
let objeto = new Object();
```

# Criação de Objetos

javascript

```
// Criação do objeto 'pessoa' usando o construtor Object()
let pessoa = new Object();

// Definindo propriedades para o objeto 'pessoa'
pessoa.nome = "Sandra";
pessoa.idade = 22;
pessoa.job = "Analista de Software";

// Definindo um método para o objeto 'pessoa'
pessoa.exibeNome = function() {
    console.log(this.nome);
};

// Usando o método 'exibeNome' para exibir o nome da pessoa
pessoa.exibeNome(); // Saída: Sandra
```

# Como os objetos são alterados?



# Alterando propriedades existentes

- ❖ Em **JavaScript**, as **propriedades** de um **objeto** podem ser alteradas diretamente através de sua referência;
- ❖ Se você já tem um objeto com algumas propriedades definidas, pode-se alterar o valor de uma propriedade existente simplesmente **atribuindo-se** um novo valor a ela.

# Alterando propriedades existentes

javascript

```
let pessoa = {  
  nome: "João",  
  idade: 30  
};  
  
console.log(pessoa.nome); // Saída: "João"  
  
// Alterando a propriedade 'nome'  
pessoa.nome = "Carlos";  
console.log(pessoa.nome); // Saída: "Carlos"
```

# Como incluir nova propriedade ?





# Adicionando nova propriedade

- ❖ Se a propriedade **não** existir no objeto, atribuir um valor a um nome de propriedade irá **criar** essa propriedade no objeto.

javascript

```
console.log(pessoa.job); // Saída: undefined

// Adicionando uma nova propriedade 'job'
pessoa.job = "Engenheiro";
console.log(pessoa.job); // Saída: "Engenheiro"
```

# Usando-se a notação []

- ❖ Além da notação de ponto, pode-se também usar a notação **colchetes** para se acessar e modificar propriedades;

javascript

```
let chave = "idade";  
pessoa[chave] = 35;  
console.log(pessoa.idade); // Saída: 35
```



Como se remover propriedades de um objeto ?

# Removendo-se propriedades de um objeto

- ❖ Pode-se **remover** uma propriedade de um objeto por meio do uso da instrução “**delete**”.

javascript

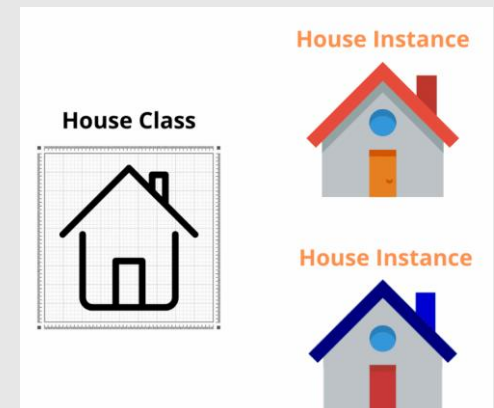
```
delete pessoa.idade;  
console.log(pessoa.idade); // Saída: undefined
```



É possível criar-se objetos  
com classes ?

# Criando-se objetos com classes

- ❖ A liberação da sintaxe de **'class'** no **ECMAScript 6** (também conhecido como **ES6** ou **ES2015**) trouxe uma forma semelhante à outras linguagens de programação para se criar objetos com base em **classes**;
- ❖ Isso pode auxiliar o aprendizado de desenvolvedores que vêm de outras linguagens.





# Criando-se objetos com classes

javascript

```
class Pessoa {  
    constructor(nome, idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
  
    apresentar() {  
        console.log(`Meu nome é ${this.nome} e tenho ${this.idade} anos.`);  
    }  
}  
  
// Criando um objeto a partir da classe Pessoa  
const joao = new Pessoa('João', 30);  
joao.apresentar(); // Exibe: "Meu nome é João e tenho 30 anos."
```

# Criando-se objetos com classes

```

javascript

class Pessoa {
  constructor(nome, idade) {
    this.nome = nome;
    this.idade = idade;
  }

  apresentar() {
    console.log(`Meu nome é ${this.nome} e tenho ${this.idade} anos.`);
  }
}

// Criando um objeto a partir da classe Pessoa
const joao = new Pessoa('João', 30);
joao.apresentar(); // Exibe: "Meu nome é João e tenho 30 anos."

```

1. Definimos uma classe chamada **Pessoa** com um construtor e um método **apresentar**.
2. O construtor é um método especial que é automaticamente chamado quando criamos uma nova instância da classe usando a palavra-chave **new**.
3. A instância **joao** é um objeto criado a partir da classe **Pessoa**.

A sintaxe de **class** oferece uma maneira mais estruturada e compreensível de criar e trabalhar com objetos, especialmente para quem já tem experiência com programação orientada a objetos em outras linguagens.

# O que é melhor ? Criar objetos com class ou com object() ?



# O que é melhor ? Criar objetos com class ou com object() ?

- ❖ A decisão entre criar objetos usando classes ou por meio do construtor **Object()** em **JavaScript** depende das necessidades específicas e da natureza do projeto.



# Construtor object()

## Vantagens:

- **Simplicidade:** Para criar objetos simples sem muita lógica ou métodos associados, o construtor `Object()` ou literais de objeto (`{}`) podem ser rápidos e simples.
- **Flexibilidade:** O JavaScript é uma linguagem dinâmica, e criar objetos com o construtor `Object()` (ou literais de objeto) permite adicionar, modificar ou remover propriedades e métodos a qualquer momento.
- **Familiaridade para Desenvolvedores Veteranos:** Desenvolvedores que estão acostumados com o estilo prototípico mais antigo de JavaScript podem se sentir mais confortáveis com essa abordagem.

# Construtor object()

## Desvantagens:

- **Menos Estruturado:** Sem uma abordagem deliberada, pode ser mais fácil criar um código desorganizado ou difícil de manter.
- **Herança:** Embora possível, a herança baseada em protótipos pode ser menos intuitiva para novos desenvolvedores do que a herança baseada em classes.



# Classes

## Vantagens:

- **Sintaxe Clara:** A sintaxe de classe, introduzida no ES6, é mais clara e semelhante à utilizada em muitas outras linguagens de programação orientadas a objetos. Isso pode facilitar para desenvolvedores que vêm de outras linguagens.
- **Estrutura e Organização:** As classes oferecem uma maneira mais estruturada de organizar e encapsular código, o que pode ser benéfico para projetos maiores.
- **Herança Simplificada:** Com a palavra-chave `extends`, as classes tornam a herança mais intuitiva e menos propensa a erros em comparação com o sistema prototípico tradicional.
- **Encapsulamento:** As classes no JavaScript mais recente permitem campos privados e métodos, oferecendo um nível de encapsulamento mais forte.

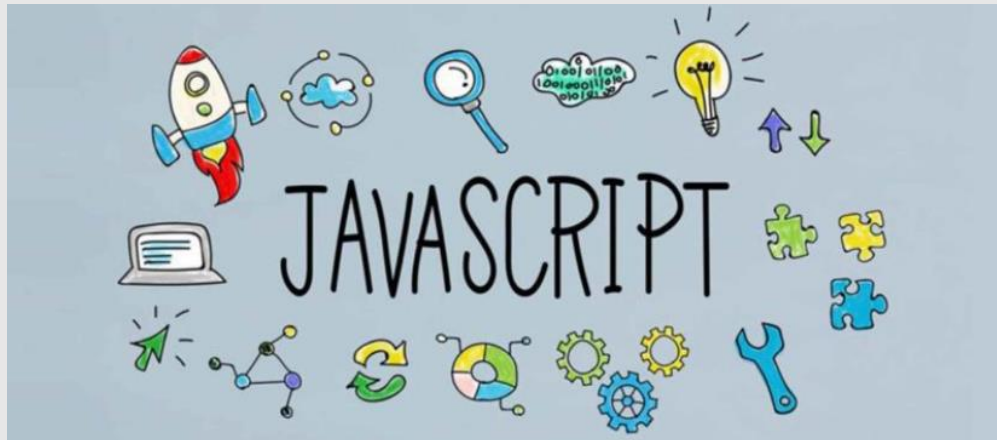
# Classes

Desvantagens:

- **Complexidade:** Para projetos ou tarefas menores, usar classes pode parecer uma abordagem exagerada ou desnecessariamente complexa.

# Coleções em JavaScript

- ❖ Em **JavaScript**, uma "**coleção**" é geralmente entendida como um objeto ou estrutura de dados que permite armazenar múltiplos valores;
- ❖ Coleções em **JavaScript**: **Arrays**, **Objetos**, **Maps** e **Sets**.



# Array em JavaScript

Um array é uma lista ordenada de valores.

Criando Arrays:

javascript

```
let frutas = ['maçã', 'banana', 'laranja'];  
let numeros = [1, 2, 3, 4, 5];
```

Acessando elementos:

javascript

```
console.log(frutas[0]); // maçã
```

# Array em JavaScript

Métodos úteis:

- `push()`: Adicionar um elemento ao final.
- `pop()`: Remover o último elemento.
- `shift()`: Remover o primeiro elemento.
- `unshift()`: Adicionar um elemento no início.
- `slice()`: Retorna uma cópia do array.
- `splice()`: Adicionar ou remover elementos de um array.
- `forEach()`: Itera sobre os elementos.

# Objetos em JavaScript

Criando Objetos:

javascript

```
let pessoa = {  
  nome: 'João',  
  idade: 30,  
  profissao: 'Engenheiro'  
};
```

Acessando propriedades:

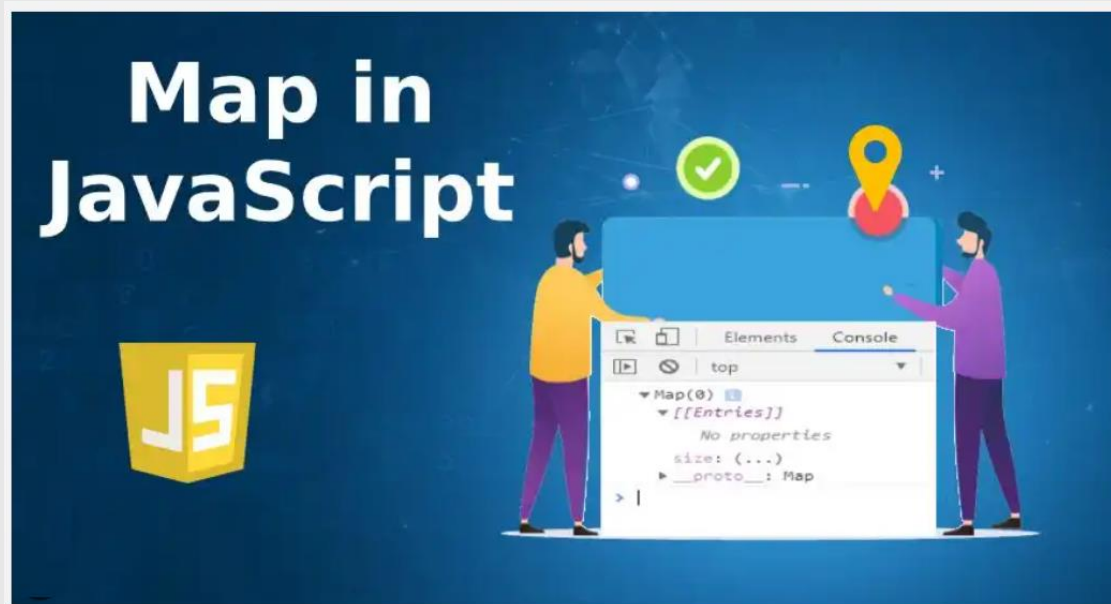
javascript

```
console.log(pessoa.nome); // João  
console.log(pessoa['idade']); // 30
```



# Maps em JavaScript

- ❖ O objeto **Map** é uma coleção simples de pares **chave/valor**;
- ❖ Qualquer valor (tanto objeto quanto valor primitivo) pode ser usado como chave ou valor.



# Criando-se um map

- ❖ Pode-se criar um **Map** vazio ou inicializá-lo com pares **chave/valor**;

javascript

```
let vazio = new Map();

let mapaInicial = new Map([
  ['chave1', 'valor1'],
  ['chave2', 'valor2']
]);
```

# Adicionando-se elementos em um map

javascript

```
let mapa = new Map();  
mapa.set('nome', 'João');  
mapa.set('idade', 30);
```

# Acessando elementos de um map

javascript

```
console.log(mapa.get('nome')); // João  
console.log(mapa.get('idade')); // 30
```

# Checando se a chave está presente

javascript

```
console.log(mapa.has('nome')); // true  
console.log(mapa.has('altura')); // false
```

# Removendo elementos de um map

javascript

```
mapa.delete('nome');  
console.log(mapa.has('nome')); // false
```

# Obtendo o tamanho de um map

javascript

```
console.log(mapa.size); // 1
```

# Limpando todos os elementos de um map

javascript

```
mapa.clear();  
console.log(mapa.size); // 0
```



# Iterando os elementos de um map

```
let mapaIterar = new Map([
  ['chave1', 'valor1'],
  ['chave2', 'valor2'],
  ['chave3', 'valor3']
]);

// Iterando pelas chaves
for (let chave of mapaIterar.keys()) {
  console.log(chave);
}
```

```
// Iterando pelos valores
for (let valor of mapaIterar.values()) {
  console.log(valor);
}

// Iterando pelos pares chave/valor
for (let [chave, valor] of mapaIterar.entries()) {
  console.log(chave, valor);
}
```

# Convertendo map para array

javascript

```
const chavesArray = [...mapaIterar.keys()];  
const valoresArray = [...mapaIterar.values()];  
console.log(chavesArray); // ['chave1', 'chave2', 'chave3']  
console.log(valoresArray); // ['valor1', 'valor2', 'valor3']
```

# Armazenando funções ou objetos como valores

javascript

```
let funcoes = new Map();
funcoes.set('imprimir', function() {
    console.log('Olá, mundo!');
});
funcoes.get('imprimir')(); // Olá, mundo!

let objetos = new Map();
objetos.set('pessoa', { nome: 'Ana', idade: 25 });
console.log(objetos.get('pessoa').nome); // Ana
```

# Exemplos – Maps

## 1. Criando um Map

Você pode criar um `Map` vazio ou inicializá-lo com pares chave/valor:

javascript

```
let vazio = new Map();
```

```
let mapaInicial = new Map([  
  ['chave1', 'valor1'],  
  ['chave2', 'valor2']  
]);
```

# Exemplos – Maps

## 2. Adicionando elementos

javascript

```
let mapa = new Map();  
mapa.set('nome', 'João');  
mapa.set('idade', 30);
```

# Exemplos – Maps

## 3. Acessando elementos

javascript

```
console.log(mapa.get('nome')); // João  
console.log(mapa.get('idade')); // 30
```

# Exemplos – Maps

## 4. Checando se uma chave está presente

javascript

```
console.log(mapa.has('nome')); // true  
console.log(mapa.has('altura')); // false
```

# Exemplos – Maps

## 5. Removendo elementos

javascript

```
mapa.delete('nome');  
console.log(mapa.has('nome')); // false
```



# Exemplos – Maps

## 6. Obtendo o tamanho do Map

javascript

```
console.log(mapa.size); // 1
```

# Exemplos – Maps

## 7. Limpando todos os elementos

javascript

```
mapa.clear();  
console.log(mapa.size); // 0
```

# Exemplos – Maps

## 8. Iterando sobre Maps

javascript

```
let mapaIterar = new Map([
  ['chave1', 'valor1'],
  ['chave2', 'valor2'],
  ['chave3', 'valor3']
]);

// Iterando pelas chaves
for (let chave of mapaIterar.keys()) {
  console.log(chave);
}

// Iterando pelos valores
for (let valor of mapaIterar.values()) {
  console.log(valor);
}

// Iterando pelos pares chave/valor
for (let [chave, valor] of mapaIterar.entries()) {
  console.log(chave, valor);
}
```

# Exemplos – Maps

## 9. Convertendo Map para Array

javascript

```
const chavesArray = [...mapaIterar.keys()];  
const valoresArray = [...mapaIterar.values()];  
console.log(chavesArray); // ['chave1', 'chave2', 'chave3']  
console.log(valoresArray); // ['valor1', 'valor2', 'valor3']
```

# Exemplos – Maps

## 10. Uso avançado: Armazenando funções ou objetos como valores

javascript

```
let funcoes = new Map();
funcoes.set('imprimir', function() {
    console.log('Olá, mundo!');
});
funcoes.get('imprimir')(); // Olá, mundo!

let objetos = new Map();
objetos.set('pessoa', { nome: 'Ana', idade: 25 });
console.log(objetos.get('pessoa').nome); // Ana
```

# Sets



# Sets

- ❖ Sets em **JavaScript** representam uma coleção de valores nos quais cada valor deve ser único;
- ❖ Isso significa que o mesmo valor **não** pode ocorrer mais de uma vez no set;
- ❖ Eles são especialmente úteis quando se deseja armazenar uma coleção de itens, mas deseja-se garantir que **cada item apareça apenas uma vez**.



# Criação de um Set

javascript

```
let frutas = new Set();  
  
frutas.add('maçã');  
frutas.add('banana');  
frutas.add('laranja');  
  
console.log(frutas); // Set(3) {"maçã", "banana", "laranja"}
```



# Adicionando valores

javascript

```
frutas.add('pêssego');  
console.log(frutas); // Set(4) {"maçã", "banana", "laranja", "pêssego"}
```

# Adicionando valores


- ❖ Ao se tentar adicionar um valor duplicado não haverá retorno de erro, simplesmente será ignorado.

javascript



```
frutas.add('pêssego');  
console.log(frutas); // Set(4) {"maçã", "banana", "laranja", "pêssego"}
```

javascript



```
frutas.add('maçã');  
console.log(frutas); // Set(4) {"maçã", "banana", "laranja", "pêssego"}
```



Vê!!! Como então tratar o erro referente à inclusão de valor duplicado no set ?

# Insert com valor duplicado

Em JavaScript, a estrutura de dados `Set` armazena apenas valores únicos, e não funciona com pares chave-valor como o objeto `Map`. Ao tentar adicionar um valor que já existe em um `Set`, simplesmente nada acontece; o valor não é adicionado novamente e nenhum erro é lançado.

Dito isso, se você quiser tratar explicitamente a tentativa de inserção de um valor duplicado em um `Set`, você pode fazer isso verificando primeiro se o valor já está presente no `Set`.

# Insert com valor duplicado

javascript

```
let frutas = new Set();

frutas.add('maçã');

if (frutas.has('maçã')) {
    console.error("Erro: A fruta 'maçã' já existe no conjunto!");
} else {
    frutas.add('maçã');
}
```

O código acima verificará se a fruta 'maçã' já existe no `Set`. Se já existir, ele exibirá uma mensagem de erro no console.

# Verificando se um valor está no set

javascript

```
console.log(frutas.has('banana')); // true  
console.log(frutas.has('uva'));    // false
```

# Removendo valores do set

javascript

```
frutas.delete('banana');  
console.log(frutas); // Set(3) {"maçã", "laranja", "pêssego"}
```

# Obtendo o tamanho do set

javascript

```
console.log(frutas.size); // 3
```



# Iterando sobre um set

Usando ``for...of``:

javascript

```
for (let fruta of frutas) {  
    console.log(fruta);  
}
```

```
// maçã  
// laranja  
// pêssego
```

# Iterando sobre um set

Usando método `forEach`:

javascript

```
frutas.forEach(fruta => {  
    console.log(fruta);  
});
```

```
// maçã  
// laranja  
// pêssego
```

# Convertendo um set em array

javascript

```
let arrayFrutas = [...frutas];  
console.log(arrayFrutas); // ["maçã", "laranja", "pêssego"]
```

# Criando-se um set a partir de um array

Se você tiver um array e quiser remover duplicatas, poderá usar um `Set`:

javascript

```
let numeros = [1, 2, 3, 4, 5, 5, 6, 6, 7];  
let numerosUnicos = new Set(numeros);  
console.log([...numerosUnicos]); // [1, 2, 3, 4, 5, 6, 7]
```

# Limpando todos os itens de um set

javascript

```
frutas.clear();  
console.log(frutas); // Set(0) {}
```