

Programação Orientada a Objetos

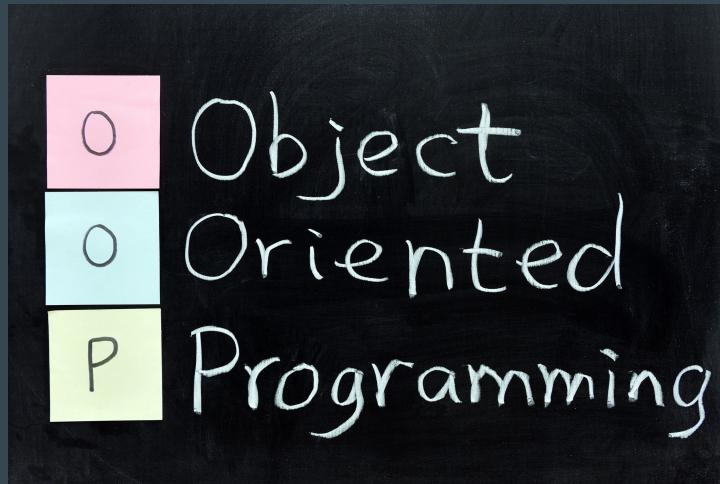
...

M04 - Objetos

M04 - Objetos

1. Objetos

- a. Criação de objetos
- b. Propriedades
- c. Métodos
- d. Iteração de objetos



M04 - Objetos

1. Objetos > Criação de objetos

M04 - Objetos

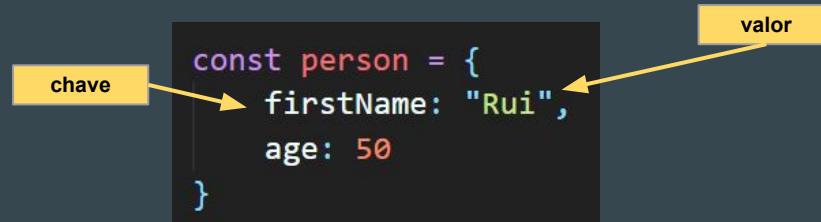
1. Objetos > Criação de objetos

- Até agora só vimos tipos de dados primitivos
 - strings ("Rui Silva")
 - números (3.14)
 - booleanos (true, false)
 - null e undefined
- Um **objeto**
 - um tipo de dados complexo
 - representa uma instância de uma entidade a modelar
 - contém um conjunto de pares de chave-valor

M04 - Objetos

1. Objetos > Criação de objetos

- O conteúdo de um objeto é composto por **propriedades** (separadas por vírgulas)
- As propriedades consistem num par **chave:valor**
 - **chaves** devem ser strings ou símbolos
 - **valores** podem ser de qualquer tipo (incluindo funções, arrays ou outros objetos)



```
const person = {
  firstName: "Rui",
  age: 50
}
```

- Por exemplo, a chave `firstName` tem o valor “Rui”
- Objetos podem ser vazios

```
const person = {}
```

M04 - Objetos

1. Objetos > Criação de objetos

- Comparação de Objetos
 - Em JavaScript, os objetos são um **tipo de referência**
 - Dois objetos distintos nunca são iguais, mesmo com as mesmas propriedades
 - Eles apontam para um endereço de memória completamente diferente
 - Objetos que partilham uma referência comum são verdadeiros na comparação

```
const num = 2
const str = "2"

console.log(num == str)      // true
console.log(num === str)    // false
```

```
const obj1 = {name: "Rui"}
const obj2 = {name: "Rui"}

console.log(obj1 == obj2)    // false
console.log(obj1 === obj2)  // false
```

```
const obj1 = {name: "Rui"}
const obj2 = obj1

console.log(obj1 == obj2)    // true
console.log(obj1 === obj2)  // true
```

M04 - Objetos

1. Objetos > Criação de objetos

- Existem várias formas para criar objetos:
 - a. criar um único objeto através de um literal objeto
 - b. criar um único objeto através da palavra-chave new
 - c. definir um construtor de objeto, e depois criar objetos do tipo do construtor
 - d. usar **classes** (estudadas mais à frente)

M04 - Objetos

1. Objetos > Criação de objetos

- Criar um literal objeto
 - lista de pares **chave:valor** dentro de {}
 - simples e legível
 - possibilidade de criação do objeto numa única declaração

```
const person = {firstName: "Rui", lastName: "Silva", age: 50, eyeColor: "azul" }
// OU
const person = {
  firstName : "Rui",
  lastName : "Silva",
  age : 50,
  eyeColor : "azul"
}
```

M04 - Objetos

1. Objetos > Criação de objetos

- Criar um literal objeto
 - baseado em variáveis
 - Ou o inverso (**desestruturação**)
 - quebra da estrutura de um objeto
 - pode-se extrair dados de arrays ou objetos em variáveis distintas

```
const firstName = "Rui"
const age = 50

const person = { firstName, age }
```

```
const emp = {name: "João", age: 22}
const {name, age} = emp

console.log(name)    // João
console.log(age)     // 22
```

M04 - Objetos

1. Objetos > Propriedades

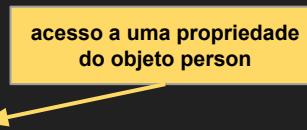
M04 - Objetos

1. Objetos > Propriedades

- Um objeto JavaScript é uma coleção de propriedades desordenadas
- As propriedades geralmente pode ser adicionadas, alteradas e removidas
- Sintaxe: `objeto.propriedade`

```
const person = {  
    firstName : "Rui",  
    lastName : "Silva"  
}  
  
console.log(person.lastName) // Silva
```

acesso a uma propriedade
do objeto person



M04 - Objetos

1. Objetos > Propriedades

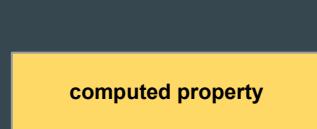
- Sintaxes alternativas:
 - objeto.propriedade
 - objeto["propriedade"]
 - objeto[expressão]

```
const person = {  
    firstName : "Rui",  
    lastName : "Silva"  
}  
  
console.log(person.lastName)  
  
console.log(person["lastName"])  
  
const x = "lastName"  
console.log(person[x])
```

M04 - Objetos

1. Objetos > Propriedades

- Propriedades calculadas (computed properties):
 - Definição de nomes de propriedades à custa do valor de uma variável



```
const feature = "color"
const car = {
  plate : "12-SA-23",
  [feature] : "azul"
}

console.log(car.color) // azul
```

M04 - Objetos

1. Objetos > Propriedades

- Iteração:
 - A declaração `for ... in` percorre as propriedades de um objeto
 - O nº de iterações do ciclo é igual ao nº de propriedades
 - Existem outras técnicas, mas esta é a mais rápida!

```
const person = {  
    firstName: "Rui",  
    lastName: "Silva",  
    age: 50  
}  
  
let text = ""  
for (let prop in person) {  
    text += `nome: ${prop} valor: ${person[prop]} \n`  
}  
  
console.log(text)  
  
/*  
 nome: firstName valor: Rui  
 nome: LastName valor: Silva  
 nome: age valor: 50  
 */
```

iteração sobre todas as propriedades do objeto person

M04 - Objetos

1. Objetos > Propriedades

- Adição de propriedades:
 - Pode-se adicionar novas propriedades a um objeto existente, basta dar-lhe um valor

```
const person = {  
    firstName: "Rui",  
    lastName: "Silva",  
    age: 50  
}  
  
person.city = "porto"  
  
console.log(person.city) // porto
```

Pode parecer que esta linha causaria um erro, mas não há nenhum problema. Isso é porque **const** contém uma **referência** para o objeto **user**. A linha faz alterações dentro do objeto, mas não mexe na referência.

M04 - Objetos

1. Objetos > Propriedades

- Remoção de propriedades:
 - A palavra-chave `delete` elimina uma propriedade de um objeto

```
const person = {firstName : "Rui", age : 50}
delete person.age
console.log(person.age) // undefined
```

- Após a remoção, a propriedade não pode ser utilizada antes de ser adicionada novamente

M04 - Objetos

1. Objetos > Métodos

M04 - Objetos

1. Objetos > Métodos

- Métodos são ações que podem ser executadas em objetos
- Um método JavaScript é uma propriedade que contém uma definição de função

```
const person = {
  firstName: "Rui",
  lastName: "Silva",
  fullName: function () {
    return `${this.firstName} ${this.lastName}`
  }
}

// Invocação do método
console.log(person.fullName()) // Rui Silva
```

Sintaxe mais abreviada

```
fullName() {
  return `${this.firstName} ${this.lastName}`
}
```

M04 - Objetos

1. Objetos > Métodos

- Palavra reservada `this`

```
const person = {  
    firstName: "Rui",  
    lastName: "Silva",  
    fullName: function () {  
        return `${this.firstName} ${this.lastName}`  
    }  
}  
  
// Invocação do método  
console.log(person.fullName()) // Rui Silva
```

Para aceder ao próprio objeto, um método pode usar a palavra-chave `this`

M04 - Objetos

1. Objetos > Iteração

M04 - Objetos

1. Objetos > Iteração

- Os objetos permitem armazenar coleções de valores com chave
- Mas muitas vezes precisamos de uma coleção ordenada, onde temos um 1º, um 2º, um 3º elemento e assim por diante. Por exemplo, precisamos disso para armazenar uma lista de algo: utilizadores, artigos, elementos HTML etc.
- Não é conveniente usar objetos, pois não fornecem métodos para gerir a ordem dos elementos
- Existe uma estrutura de dados especial chamada **Array**, para armazenar coleções ordenadas

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Declaração
 - Duas formas de sintaxe para criar um array vazio

```
let arr = new Array();
let arr = [];
```

- A segunda sintaxe é mais usada. Podemos também fornecer elementos iniciais:

```
let fruits = ["Apple", "Orange", "Plum"];
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Acesso
 - Os elementos do array são numerados (índices), começando pelo índice zero.
 - Podemos obter um elemento colocando o seu índice entre chavetas retas:

```
let fruits = ["Apple", "Orange", "Plum"];

alert(fruits[0]); // Apple
alert(fruits[1]); // Orange
alert(fruits[2]); // Plum
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Transformações
 - Podemos alterar um elemento dando o seu índice e atribuindo um novo valor

```
let fruits = ["Apple", "Orange", "Plum"];
fruits[2] = 'Pear'; // ["Apple", "Orange", "Pear"]
```

- Podemos adicionar um novo elemento dando um índice novo e atribuindo um novo valor

```
fruits[3] = 'Lemon'; // now ["Apple", "Orange", "Pear", "Lemon"]
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Contagem
 - O nº total de elementos de um array é dado pela propriedade `length`

```
let fruits = ["Apple", "Orange", "Plum"];
alert(fruits.length); // 3
```

- Podemos usar o `alert` para mostrar o array todo

```
let fruits = ["Apple", "Orange", "Plum"];
alert(fruits); // Apple,Orange,Plum
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Contagem
 - Para ser preciso, a propriedade `length` não é a contagem de valores no array, mas devolve o maior índice numérico (mais um)
 - Por exemplo:

```
let fruits = [];
fruits[123] = "Apple";

alert(fruits.length); // 124
```

- Os “buracos” gerados são preenchidos com o valor `undefined`

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Contagem
 - A propriedade `length` não é só de leitura, na realidade é também usada para truncar ou limpar o array

```
let arr = [1, 2, 3, 4, 5];

arr.length = 2; // trunca o array para 2 elementos
alert(arr); // [1, 2]

arr.length = 5;
alert(arr[3]); // undefined: os valores não são recuperados!
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Referência
 - Lembrem-se, que existem apenas 7 tipos básicos em JavaScript
 - Um array é um tipo especial de objeto e, portanto, comporta-se como um objeto
 - Os parêntesis retos usados para aceder a uma propriedade arr[0] vêm realmente da sintaxe do objeto, sendo o mesmo que obj[key], em que arr é o objeto, enquanto que números são usados como chaves
 - Os arrays estendem os objetos fornecendo métodos especiais para trabalhar com coleções ordenadas de dados e também com a propriedade **length**

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Referência
 - Por exemplo, um array (tal como um objeto) é copiado por referência

```
let fruits = ["Banana"]

let arr = fruits; // cópia por referência (duas variáveis referenciam o mesmo array)

alert(arr === fruits); // true

arr.push("Pear"); // modifica o array por referência

alert(fruits); // Banana, Pear - 2 elementos agora
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Iteração
 - Uma das formas mais antigas de iterar sobre elementos de um array é com o ciclo **for** que percorre o array usando os seus índices:

```
let arr = ["Apple", "Orange", "Pear"];  
  
for (let i = 0; i < arr.length; i++) {  
    alert(arr[i]);  
}
```

i - posição (índice)
arr[i] - elemento

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Iteração (for..of)
 - Outra forma popular é usar o ciclo for..of
 - Neste caso, dentro do ciclo, só se consegue aceder aos elementos do array

```
let fruits = ["Apple", "Orange", "Plum"];  
  
// itera sobre os elementos do array  
for (let fruit of fruits) {  
    alert(fruit);  
}
```

- O for..of não dá acesso à posição (índice) do elemento atual, apenas ao seu valor, mas na maioria dos casos é o suficiente. E é mais curto.

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Métodos sobre arrays:

Adicionar / Remover elementos

Pesquisar elementos

Iterar sobre elementos

Transformar o array

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Métodos sobre arrays:

Adicionar / Remover elementos

Pesquisar elementos

Iterar sobre elementos

Transformar o array

M04 - Objetos

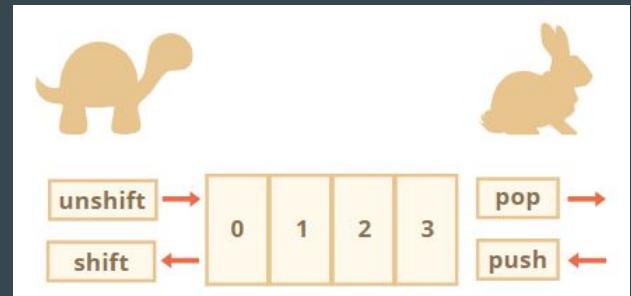
1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:
 - `push(...elems)` - adiciona elementos ao final
 - `pop()` - extrai um elemento do final
 - `shift()` - extrai um elemento do início
 - `unshift(...elems)` - adiciona elementos ao início
 - `splice(pos, delCount, ...elems)` - no índice `pos`, exclui `delCount` elementos e insere `elems`
 - `slice(init, end)` - cria novo array e copia elementos da posição `init` até `end` para ele
 - `concat(...elems)` - retorna novo array: copia os membros do atual e adiciona `elems` a ele

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:
 - `push(...elems)` - adiciona elementos ao final
 - `pop()` - extrai um elemento do final
 - `shift()` - extrai um elemento do início
 - `unshift(...elems)` - adiciona elementos ao início



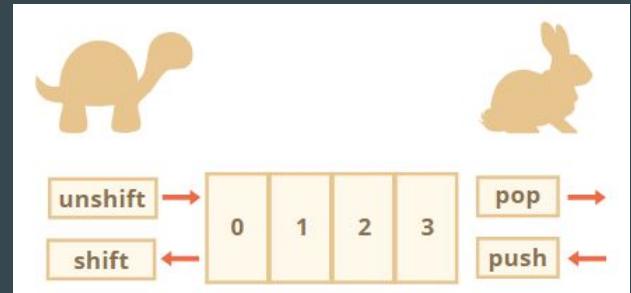
M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:

- `push(...elems)`

- adiciona elementos ao final
 - retorna o tamanho atual (`length`) do array



```
let fruits = ["Apple", "Orange"];
fruits.push("Pear", "Banana");
alert(fruits); // Apple, Orange, Pear, Banana
```

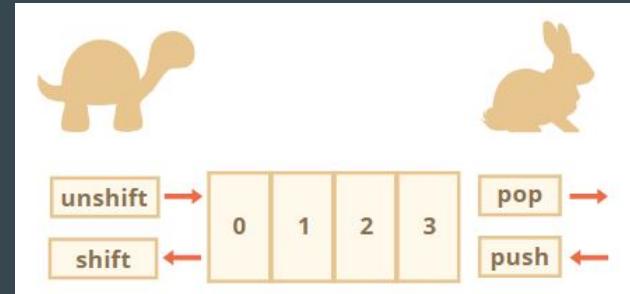
M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:

- **pop()**

- extrai um elemento do final do array
 - retorna o elemento removido



```
let fruits = ["Apple", "Orange", "Pear"];

alert(fruits.pop()); // remove "Pear" e mostra-o numa janela de alerta

alert(fruits); // Apple, Orange
```

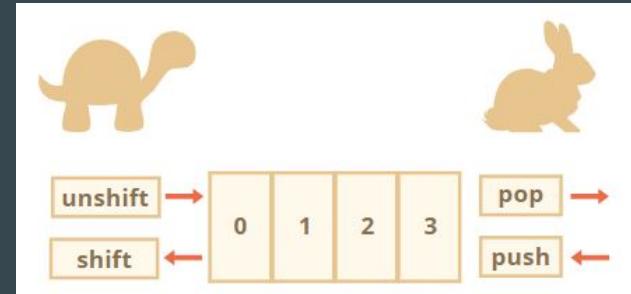
M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:

- shift()

- extrai um elemento do início do array
- retorna o elemento removido



```
let fruits = ["Apple", "Orange", "Pear"];

alert(fruits.shift()); // remove Apple e exibe-o numa janela de alerta

alert(fruits); // Orange, Pear
```

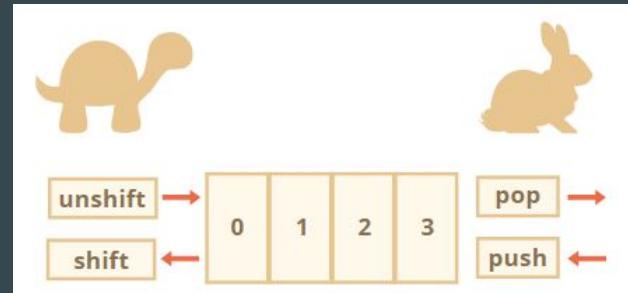
M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:

- `unshift(...elems)`

- adiciona elementos ao início
 - retorna o tamanho atual (`length`) do array



```
let fruits = ["Orange", "Pear"];

fruits.unshift('Apple');

alert(fruits); // Apple, Orange, Pear
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:
 - Como excluir um elemento interior de um array?
 - Os arrays são objetos, por isso pode usar **delete**:

```
let arr = ["I", "go", "home"];

delete arr[1]; // remove "go"

alert(arr[1]); // undefined
alert(arr); // I, ,home
alert(arr.length); // 3
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:
 - Solução para evitar o problema anterior da geração de “buracos”, use o método **splice**:
 - **splice(pos, delCount, ...elems)** - no índice **pos**, exclui **delCount** elementos e insere **elems**

```
let arr = ["I", "study", "JavaScript"];

arr.splice(1, 1); // do index 1 remove 1 elemento

alert(arr); // ["I", "JavaScript"]
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:
 - `splice(pos, delCount, ...elems)` - no índice `pos`, exclui `delCount` elementos e insere `elems`
 - Pode também usar o método para remover e inserir elementos em simultâneo:

```
let arr = ["I", "study", "JavaScript", "right", "now"];

// remove os 3 primeiros elementos e insere os restantes
arr.splice(0, 3, "Let's", "dance");

alert(arr) // now ["Let's", "dance", "right", "now"]
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:
 - `slice(init, end)` - cria um novo array e copia elementos da posição `init` até `end` (não inclusive) para ele

```
let arr = ["t", "e", "s", "t"];
alert(arr.slice(1, 3)); // e,s (copia de 1 a 3)
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Adicionar / Remover elementos:
 - concat(...elems)
 - copia todos os membros do atual e adiciona **elems** a ele. Se algum dos elementos for um array, os seus elementos serão utilizados
 - retorna um novo array

```
let arr = [1, 2];

// cria um array de: arr e [3,4]
alert(arr.concat([3, 4])); // 1,2,3,4

// cria um array de: arr e [3,4] e [5,6]
alert(arr.concat([3, 4], [5, 6])); // 1,2,3,4,5,6

// cria um array de: arr e [3,4], a seguir adiciona os valores 5 e 6
alert(arr.concat([3, 4], 5, 6)); // 1,2,3,4,5,6
```

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Métodos sobre arrays:

Adicionar / Remover elementos

Pesquisar elementos

Iterar sobre elementos

Transformar o array

M04 - Objetos

1. Objetos > Iteração (Arrays)

- Iterar sobre elementos:
 - O método `forEach` permite executar uma função para cada elemento do array
 - Sintaxe:

```
arr.forEach(function (item, index, array) {  
    // ... fazer alguma coisa com item  
});
```

- Exemplo:
- ```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
 alert(`#${item} está no índice ${index} no ${array}`);
});
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Métodos sobre arrays:

Adicionar / Remover elementos

Pesquisar elementos

Iterar sobre elementos

Transformar o array

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Pesquisar elementos no array:
  - `indexOf/lastIndexOf(elem, pos)` - procura o `elem` iniciando na posição `pos`, e retorna o índice ou `-1` se não for encontrado
  - `includes(value)` - retorna verdadeiro se o array tiver `value`, caso contrário, falso
  - `some(fn)` - testa se pelo menos um elemento do array passa no teste implementado pela função fornecida
  - `every(fn)` - testa se todos os elementos do array passam no teste implementado pela função fornecida
  - `find/filter(func)` - filtra os elementos através da função, retorna o primeiro/todos os valores que a fazem retornar verdadeira
  - `findIndex(func)` - é como find, mas retorna o índice em vez de um valor

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Pesquisar elementos no array:
  - `indexOf(elem, pos)` - procura o `elem` iniciando na posição `pos`, e retorna o índice ou `-1` se não for encontrado
  - `lastIndexOf(elem, pos)` - o mesmo, mas procura da direita para a esquerda.
  - `includes(value)` - retorna verdadeiro se o array tiver `value`, caso contrário, falso.

```
let arr = [1, 0, false];
alert(arr.indexOf(0)); // 1
alert(arr.indexOf(false)); // 2
alert(arr.indexOf(null)); // -1
alert(arr.lastIndexOf(0)); // 1
alert(arr.includes(1)); // true
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Pesquisar elementos no array:
  - `some(func)`
    - testa se pelo menos um elemento do array passa no teste implementado pela função fornecida
    - retorna um valor booleano

```
// verificar se existe pelo menos um número par no array
const array = [1, 2, 3, 4, 5];

// sintaxe tradicional
let res = false;
for (let i = 0; i < array.length; i++) {
 const element = array[i];
 if (element % 2 === 0) {
 res = true;
 break;
 }
}
alert(res);

// sintaxe moderna (não abreviada)
let even = array.some(
 function (element) {
 return element % 2 === 0;
 }
)
alert(even)

// sintaxe moderna (abreviada)
alert(array.some(element => element % 2 === 0)); // true
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Pesquisar elementos no array:
  - `every(func)`
    - testa se todos os elementos do array passam no teste da função fornecida
    - retorna um valor booleano

```
const isBelowThreshold = (currentValue) => currentValue < 40;

const array1 = [1, 30, 39, 29, 10, 13];

alert(array1.every(isBelowThreshold)); // true
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Pesquisar elementos no array:
  - `find(func)`
    - filtra os elementos através dumha função, retorna o primeiro valor que a fazem retornar verdadeira
    - Se não encontrar devolve `undefined`
  - O método `findIndex(fn)` é similar devolvendo o índice ou `-1` caso não haja ocorrências

```
let users = [
 { id: 1, name: "John" },
 { id: 2, name: "Pete" },
 { id: 3, name: "Mary" }
];

let user = users.find(item => item.id == 1);

alert(user.name); // John
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Pesquisar elementos no array:
  - filter(func)
    - filtra os elementos com uma função
    - retorna array com todos elementos que fazem com que a função devolva verdadeiro
    - Se não encontrar devolve []

```
let users = [
 { id: 1, name: "John" },
 { id: 2, name: "Pete" },
 { id: 3, name: "Mary" }
];

// retorna array com dois utilizadores
let someUsers = users.filter(item => item.id < 3);

alert(someUsers.length); // 2
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Métodos sobre arrays:

Adicionar / Remover elementos

Pesquisar elementos

Iterar sobre elementos

Transformar o array

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `map(func)` - cria um novo array a partir dos resultados da chamada de `func` para cada elemento do array
  - `sort(func)` - ordena um array (*in place*). Usa `func` para controlar a ordenação.
  - `reverse()` - inverte o array *in place*
  - `split(sep)/join(sep)` - converte uma string em array e vice-versa baseado em `sep`.
  - `reduce(func, init)` - calcula um único valor sobre a matriz chamando `func` para cada elemento e passando um resultado intermediário entre as chamadas.
  - `fill(value, start, end)` - preenche o array com valores repetidos do início ao fim do índice.

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `map(func)`
    - Cria um novo array preenchido com os resultados da chamada de uma função fornecida em todos os elementos do array dado

```
const array1 = [1, 4, 9, 16];

const map1 = array1.map(x => x * 2);

console.log(map1); // [2, 8, 18, 32]
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `sort(func)`
    - ordena um array *in place*, alterando sua ordem dos elementos.
    - retorna o array ordenado, mas o valor retornado geralmente é ignorado, pois o array atual é modificado (**método de mutação**)

```
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months); // ["Dec", "Feb", "Jan", "March"]
```

```
const array1 = [1, 30, 4, 21, 100000];
array1.sort();
console.log(array1); // [1, 100000, 21, 30, 4]
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `sort(func)`
    - como devem ter reparado a ordenação numérica está incorreta! Porquê?
    - os elementos, por omissão, são ordenados como strings
    - para strings, a ordem lexicográfica é aplicada e, de fato, "30">> "100000".
    - para controlar a ordenação, deve-se fornecer uma função como argumento de `sort`

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `sort(func)`

- Exemplo:

```
function compareNumeric(a, b) {
 if (a > b) return 1;
 if (a == b) return 0;
 if (a < b) return -1;
}

let arr = [1, 2, 15];

arr.sort(compareNumeric);

alert(arr); // 1, 2, 15
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `reverse(func)`
    - Inverte um array *in place*, alterando a ordem dos elementos.
    - Retorna o array invertido, mas o valor retornado geralmente é ignorado, pois o array atual é modificado (**método de mutação**)

```
let arr = [1, 2, 3, 4, 5];
arr.reverse();

alert(arr); // 5,4,3,2,1
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `join(sep)`
    - Cria e retorna uma nova string concatenando todos os elementos de um array, separados por vírgulas ou uma sequência separadora (`sep`) especificada

```
const elements = ['Fire', 'Air', 'Water'];

console.log(elements.join()); // "Fire,Air,Water"

console.log(elements.join(' ')); // "FireAirWater"

console.log(elements.join('-'));// "Fire-Air-Water"
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `join(sep)`
    - O inverso de `join` é o método `split` aplicado numa string

```
const str = 'The quick brown fox jumps over the lazy dog.';

const words = str.split(' ');
console.log(words[3]); // "fox"

const chars = str.split('');
console.log(chars[8]); // "k"

const strCopy = str.split();
console.log(strCopy); // ["The quick brown fox jumps over the lazy dog."]
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `reduce(func)`
    - quando precisamos iterar sobre um array - usamos `forEach`, `for` ou `for..of`
    - quando precisamos iterar e retornar os dados para cada elemento - usamos `map`
    - o método `reduce` permite calcular um único valor com base num array
    - sintaxe:

```
let value = arr.reduce(function (accumulator, item, index, array) {
 // ...
}, [initial]);
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - **reduce(func)**

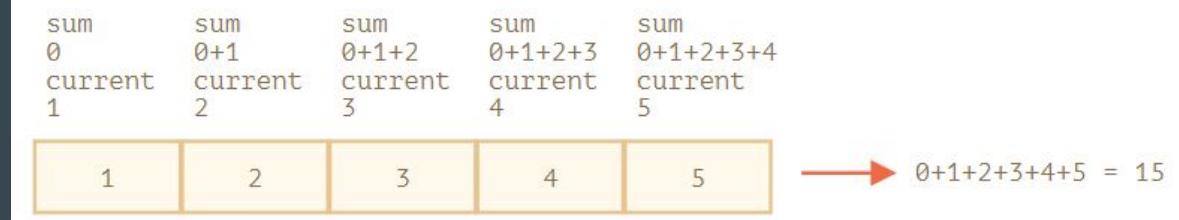
```
let value = arr.reduce(function (accumulator, item, index, array) {
 // ...
}, [initial]);
```

- a função é aplicada a todos os elementos do array, um após o outro, e "continua" o resultado na próxima chamada.
- argumentos:
  - **accumulator** - é o resultado da chamada de função anterior, igual a **initial** na primeira vez (se **initial** for fornecido)
  - **item** - é o elemento (item) atual do array
  - **index** - é a sua posição
  - **array** - é o array

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:
  - `reduce(func)`
  - exemplo:



```
let arr = [1, 2, 3, 4, 5];

let result = arr.reduce((sum, current) => sum + current, 0);

alert(result); // 15
```

# M04 - Objetos

## 1. Objetos > Iteração (Arrays)

- Transformar array:

- `fill(value[, start[, end]])`

- altera todos os elementos de um array para um valor estático, de um índice inicial (padrão 0) para um índice final (padrão `array.length`)
    - retorna o array modificado

```
const array1 = [1, 2, 3, 4];

// preenche com 0 desde a posição 2 até à posição 4
console.log(array1.fill(0, 2, 4)); // [1, 2, 0, 0]

// preenche com 5 desde a posição 1
console.log(array1.fill(5, 1)); // [1, 5, 5, 5]

console.log(array1.fill(6)); // [6, 6, 6, 6]
```

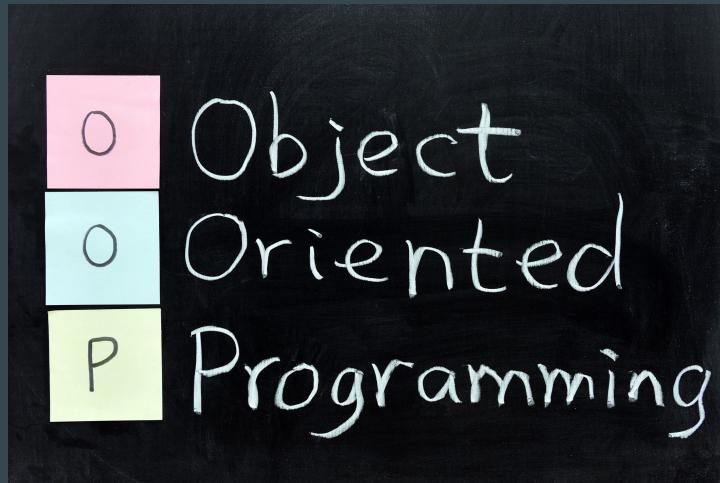
# M04 - Objetos

## 1. Objetos

- a. Criação de objetos
- b. Propriedades
- c. Métodos
- d. Iteração de objetos

## 2. Classes

- a. Criação de classes
- b. Propriedades
- c. Métodos
- d. Estáticas



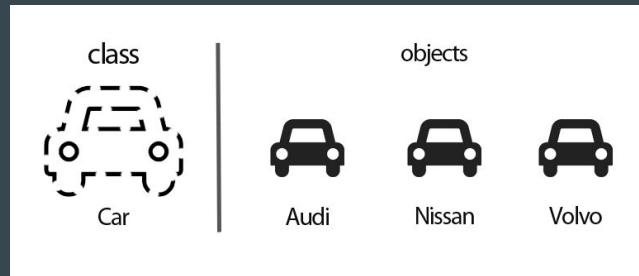
# M04 - Objetos

2. Classes > Criação

# M04 - Objetos

## 2. Classes > Criação

- Criação de objetos literais é útil quando se quer criar um único objeto para fins específicos
- Contudo, muitas vezes necessitamos de criar muitos objetos similares!



- A partir do ES2015, formaliza-se um conjunto similar de objetos através do conceito de **classes**
  - encapsulam dados e procedimentos que descrevem o conteúdo e comportamento de entidades do mundo real (ex: carro, cliente, produto), representadas por **objetos**
  - é uma descrição abstrata de um conjunto de objetos com características similares

# M04 - Objetos

## 2. Classes > Criação

- A sintaxe base de uma classe é a seguinte:

```
class MyClass {
 constructor(...){
 // ...
 }
 method1(...){ }
 method2(...){ }
 get property(...){ }
 set property(...){ }
 static staticMethod(...){ }
 // ...
}
```

# M04 - Objetos

## 2. Classes > Criação

- Para definir uma classe usa-se a keyword `class`
- A construção (instanciação) de um objeto é feita através de `new`, seguido do nome da classe
- Esta ação invoca o método `constructor`
- Podem ser passados parâmetros ao construtor

The diagram illustrates the creation of a `User` class and its instances. It consists of three parts:

- Definição da classe**: An annotation pointing to the `class User {` line, indicating the class definition.
- Criação de objetos**: An annotation pointing to the two `const` statements creating instances: `const firstUser = new User("Rui", 23)` and `const secondUser = new User("Maria", 12)`.
- Acesso a propriedades dos objetos**: An annotation pointing to the `console.log` statements outputting properties: `firstUser.name` and `secondUser.age`.

```
class User {
 constructor(name, age) {
 this.name = name
 this.age = age
 }
}

const firstUser = new User("Rui", 23)
const secondUser = new User("Maria", 12)

console.log(firstUser.name) // Rui
console.log(secondUser.age) // 12
```

# M04 - Objetos

## 2. Classes > Criação

- É possível verificar se um objeto pertence a uma classe com o operador instanceof

```
class User {
 constructor(name, age) {
 this.name = name
 this.age = age
 }
}

const firstUser = new User("Rui", 23)
const secondUser = new User("Maria", 12)

console.log(firstUser instanceof User) // true
```

# M04 - Objetos

2. Classes > Propriedades

# M04 - Objetos

## 2. Classes > Propriedades

- As classes têm propriedades que expoem características dos objetos a ela pertencentes
- As propriedades são definidas por dois grandes grupos:
  - Funções acessoras (**getters**) – uso do **get**
  - Funções modificadoras (**setters**) – uso do **set**

```
class User {
 // Getter da propriedade name
 get name() { }

 // Setter da propriedade name
 set name(value) { }
}
```

# M04 - Objetos

## 2. Classes > Propriedades

```
class User {
 // Getter da propriedade name
 get name() { }

 // Setter da propriedade name
 set name(value) { }
}

const newUser = new User()
newUser.name = "Ricardo" // chama setter
console.log(newUser.name) // chama getter
```

Chama o **setter** da propriedade **name**

Chama o **getter** da propriedade **name**

# M04 - Objetos

## 2. Classes > Propriedades

```
class User {
 constructor(name) {
 // invoca o setter
 this.name = name
 }
 get name() {
 return this._name
 }
 set name(value) {
 if (value.length < 4) {
 console.log("nome muito curto")
 return
 }
 this._name = value
 }
}
```

```
const newUser = new User("Ricardo")
console.log(newUser.name) // Ricardo
```

# M04 - Objetos

## 2. Classes > Propriedades

```
class User {
 constructor(name) { ←
 // invoca o setter
 this.name = name
 }
 get name() {
 return this._name
 }
 set name(value) {
 if (value.length < 4) {
 console.log("nome muito curto")
 return
 }
 this._name = value
 }
}
```

1

```
const newUser = new User("Ricardo")
console.log(newUser.name) // Ricardo
```

1

Instancia um novo objeto chamando o  
método constructor da classe

# M04 - Objetos

## 2. Classes > Propriedades

```
class User {
 constructor(name) { ←
 // invoca o setter
 this.name = name
 }
 get name() {
 return this._name
 }
 set name(value) {
 if (value.length < 4) {
 console.log("nome muito curto")
 return
 }
 this._name = value
 }
}
```

1

```
const newUser = new User("Ricardo")
console.log(newUser.name) // Ricardo
```

2

1

Instancia um novo objeto chamando o método constructor da classe

2

É atribuído um novo valor à propriedade name. É invocado o seu método set

# M04 - Objetos

## 2. Classes > Propriedades

```
class User {
 constructor(name) { ←
 // invoca o setter
 this.name = name
 }
 get name() {
 return this._name
 }
 set name(value) {
 if (value.length < 4) {
 console.log("nome muito curto")
 return
 }
 this._name = value
 }
}
```

1

```
const newUser = new User("Ricardo")
console.log(newUser.name) // Ricardo
```

2

1

Instancia um novo objeto chamando o método constructor da classe

2

É atribuído um novo valor à propriedade name. É invocado o seu método set

3

A variável privada \_name fica com o valor "Ricardo"

3

# M04 - Objetos

## 2. Classes > Propriedades

```
class User {
 constructor(name) { ← 1
 // invoca o setter
 this.name = name
 } ← 2
 get name() { ← 4
 return this._name
 }
 set name(value) {
 if (value.length < 4) {
 console.log("nome muito curto")
 return
 } ← 3
 this._name = value
 }
}
```

```
const newUser = new User("Ricardo")
console.log(newUser.name) // Ricardo
```

- 1 Instancia um novo objeto chamando o método constructor da classe
- 2 É atribuído um novo valor à propriedade name. É invocado o seu método set
- 3 A variável privada \_name fica com o valor "Ricardo"
- 4 É invocado o método get da propriedade name que devolve o valor da sua variável privada \_name

# M04 - Objetos

## 2. Classes > Propriedades

```
class User {
 constructor(name) {
 // invoca o setter
 this.name = name
 }
 get name() {
 return this._name
 }
 set name(value) {
 if (value.length < 4) {
 console.log("nome muito curto")
 return
 }
 this._name = value
 }
}
```

```
const newUser = new User("Ricardo")
console.log(newUser.name) // Ricardo
```

```
const newUser = new User("Eva") // nome muito curto
```

O construtor chama o método set da propriedade name. A avaliação do if é verdadeira pelo que é apresentada mensagem de erro na consola e a variável privada \_name não é alimentada

# M04 - Objetos

2. Classes > Métodos

# M04 - Objetos

## 2. Classes > Métodos

- Os métodos são definidos dentro da classe
- Não é necessário explicitamente usar a keyword `function`

The diagram illustrates the definition and invocation of a method in JavaScript. On the left, a code editor window shows a class definition for `User`. A yellow callout box labeled "Definição do método" points to the `saiHi()` method definition. On the right, another yellow callout box labeled "Invocação do método" points to the line where `newUser.saiHi()` is called in the code below. The code is as follows:

```
class User {
 constructor(name) {
 // invoca o setter
 this.name = name
 }
 saiHi() {
 return `Olá ${this.name}!`
 }
}

const newUser = new User("Ricardo")
console.log(newUser.saiHi()) // Olá Ricardo!
```

# M04 - Objetos

2. Classes > Estáticas

# M04 - Objetos

## 2. Classes > Estáticas

- Não requerem uma instância da classe
- Não podem aceder implicitamente aos dados (`this`) de uma qualquer instância
- Sintaxe:
  - Palavra-chave `static` no início da assinatura do método
  - Chamada através de: `nomeClasse.NomeMetodoEstatico()`
- Principais funções:
  - Métodos “Factory”
  - Implementar funções que pertencem à classe, mas não a nenhum objeto em particular

# M04 - Objetos

## 2. Classes > Estáticas

- Métodos “Factory”

Definição da estática

```
class Article {
 constructor(title, date) {
 this._title = title
 this._date = date
 }

 get title() {
 return this._title
 }

 static createTodays() {
 return new Article("Última Hora", new Date())
 }
}

const article = Article.createTodays()
console.log(article.title) // Última Hora
```

Invocação da estática