

voce poderia me explicar a diferença desse comando `"pizzaJson.find((item)=>item.id == cart[i].id);"` para esse comando `"pizzaJson.find((item)=>{
 return item.id == cart[i].id;
});"`



Esses dois comandos são diferentes na forma como a função de callback é escrita, mas o resultado é o mesmo. Ambos comandos usam o método **find()** do objeto **pizzaJson** para procurar um objeto de pizza que corresponda a um item específico do carrinho.

A principal diferença entre os dois comandos está na forma como a função de callback é escrita. A primeira versão usa uma expressão lambda (arrow function) para especificar a função de callback:

javascript

 Copy code

```
pizzaJson.find((item)=>item.id == cart[i].id);
```

Nesse caso, a função de callback consiste apenas em uma expressão que compara os valores do **id** do item em **pizzaJson** com o **id** do item atual em **cart**. Como a expressão contém apenas uma instrução, o valor retornado da expressão será automaticamente o valor de retorno da função de callback.

A segunda versão usa uma função de callback explícita:

javascript

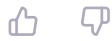
 Copy code

```
pizzaJson.find((item)={ return item.id == cart[i].id; });
```

Nesse caso, a função de callback é uma função nomeada, que é declarada usando a palavra-chave **function**. O corpo da função contém a mesma instrução que a primeira versão, mas é necessário usar a palavra-chave **return** para especificar explicitamente o valor de retorno da função.

Em resumo, a principal diferença entre as duas formas está na sintaxe utilizada para definir a função de callback. A expressão lambda é uma forma mais curta e concisa de definir funções simples em JavaScript, enquanto que a função de callback explícita é mais flexível

e pode ser usada para funções mais complexas que requerem várias instruções. No entanto, ambas as formas são válidas e produzem o mesmo resultado no contexto do método `find()`.



D

entao a diferença é que uma ja é feita direto sem precisar colocar o return e a outra declara o return ?



Sim, exatamente! A principal diferença entre as duas formas está na forma como a função de callback é escrita. Na primeira forma, usando uma expressão lambda, a função de callback é implícita e retorna automaticamente o resultado da expressão, sem a necessidade de usar a palavra-chave **return**. Na segunda forma, usando uma função de callback explícita, é necessário declarar a função usando a palavra-chave **function** e usar explicitamente a palavra-chave **return** para retornar o resultado da expressão.

Em resumo, ambas as formas são válidas e produzem o mesmo resultado no contexto do método `find()`. A escolha entre uma forma ou outra geralmente depende da preferência pessoal do programador ou das necessidades específicas do projeto. A expressão lambda pode ser mais concisa e legível para funções simples e curtas, enquanto a função de callback explícita pode ser mais flexível para funções mais complexas e longas.



D

entao sempre que eu for criar uma função que só contenha nela o comando para retornar algo eu posso fazer direto, sem colocar o return ?



Sim, exatamente! Quando você tem uma função que contém apenas uma expressão, você pode omitir a palavra-chave **return** e usar uma expressão lambda para retornar o resultado diretamente. Isso pode tornar o código mais conciso e legível.

Por exemplo, considere a seguinte função que retorna o dobro de um número:

typescript

Copy code

```
function double(number) { return number * 2; }
```

Você poderia reescrever essa função usando uma expressão lambda da seguinte forma:

typescript

 Copy code

```
const double = (number) => number * 2;
```

Observe que a expressão **number * 2** é a única instrução na função de callback. Como não há outras instruções na função, você pode omitir as chaves **{}** e a palavra-chave **return**, tornando a função mais curta e concisa.

Em resumo, quando você tem uma função que contém apenas uma expressão, você pode usar uma expressão lambda e omitir a palavra-chave **return** para tornar o código mais conciso e legível.
