

Módulo F: avançando os estudos em JS

Conteúdo do módulo F:

- Variáveis compostas;
- Uso de funções e eventos;
- Passagem de parâmetros;
- Exercícios propostos.

▼ Aula 15 - Variáveis compostas

Variáveis simples só conseguem armazenar um valor por vez.

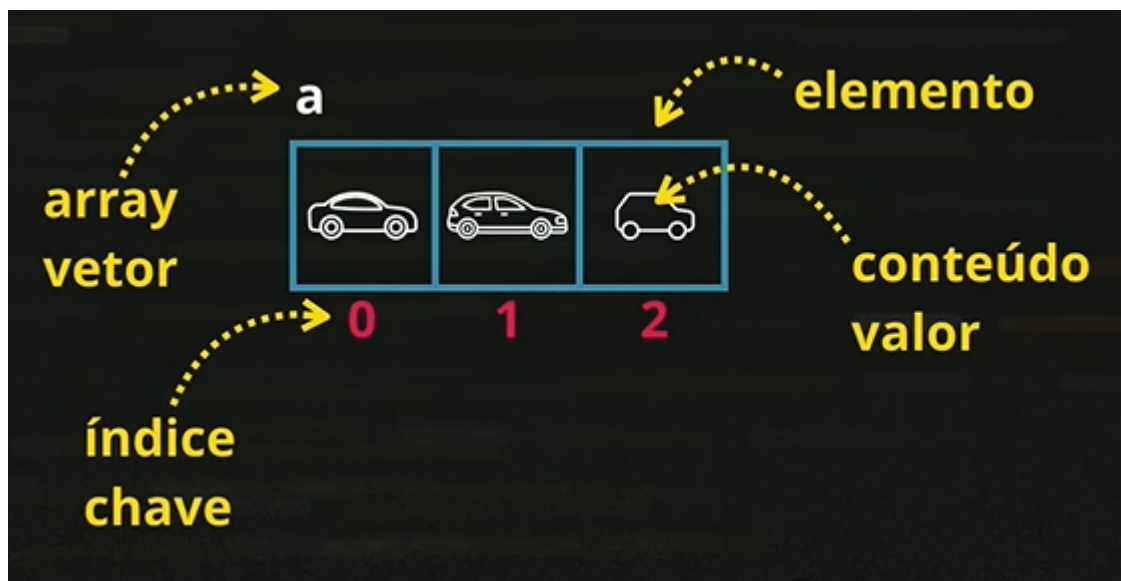
Variáveis compostas devem ser capazes de armazenar vários valores em uma mesma estrutura.

Variável composta = vetores

Vetor = Array

Todo array em JS é um objeto.

Vetores têm posições, e elas têm identificadores. Esses identificadores (índice) sempre começam com 0.



Exemplificação de um vetor. Fonte: Curso em Vídeo.

Declarando uma variável composta em JS:

```
// exemplo:  
let num = []  
let vetor = [3, 5, 7] // vetor com 3 posições e com os valores declarados  
  
// acrescentando um valor a uma determinada posição do vetor:  
vetor[3] = 6 // a posição 3 será criada, armazenando o valor 6. Número  
total de posições: 4 [0, 1, 2, 3]  
  
// adicionando um valor à última posição do vetor:  
vetor.push(7) // acréscimo do 7 ao final
```

```
// descobrindo o tamanho do vetor:
vetor.length

// ordenando os valores do vetor em ordem crescente:
vetor.sort() // resultado: [3, 5, 6, 7]

// buscando valores dentro de um vetor
vetor.indexOf(7) // procura pelo valor 7 dentro do vetor. Se não existir, in
formará -1
```

Exibindo os valores de cada posição de um vetor:

Código:

```
let num = [10, 8, 23, 15] // criação do vetor

// usando um laço de repetição para exibir os valores do vetor
for (let i = 0; i < num.length; i++) {
  console.log(`Posição ${i} : ${num[i]}`)
}
```

Resultado:

```
Info: Start process (20:50:40)
Posição 0 : 10
Posição 1 : 8
Posição 2 : 23
Posição 3 : 15
Posição 4 : 2
Posição 5 : 1
Info: End process (20:50:40)
```

Fonte: Compilação da autora.

Forma simplificada de exibir os valores das posições de um vetor:

```
for (let pos in num){ // para cada posição na variável composta num...
  console.log(`Posição ${pos} : ${num[pos]}`) // ... mostra o valor da p
```

```
    posição atual  
  }
```

Resultado da execução:

```
Info: Start process (20:56:55)  
Posição 0 : 10  
Posição 1 : 8  
Posição 2 : 23  
Posição 3 : 15  
Posição 4 : 2  
Posição 5 : 1  
Info: End process (20:56:56)
```

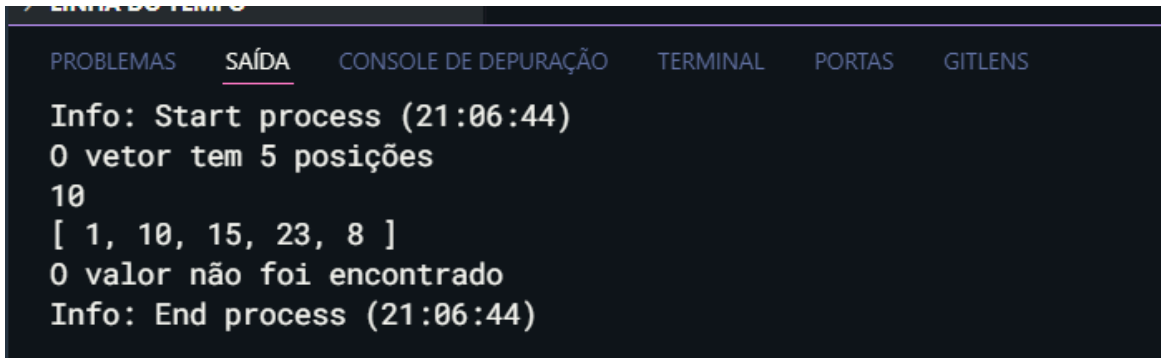
Fonte: Compilação da autora.

Procurando por um valor dentro do vetor:

Código:

```
let num = [10, 8, 23, 15] // criação do vetor  
num.push(1) // adiciona um elemento ao final do vetor  
  
console.log(`O vetor tem ${num.length} posições`) // mostra o tamanho  
do vetor  
console.log(num[0]) // mostra o valor da posição 0  
console.log(num.sort()) // Ordena em ordem crescente os valores do ve  
tor  
  
// procurando por um valor dentro do vetor  
let pos = num.indexOf(23)  
if (pos === -1) {  
  console.log('O valor não foi encontrado')  
}  
else {  
  console.log(`O valor 2 está na posição ${pos}`)  
}
```

Resultado da execução:



```
Info: Start process (21:06:44)
0 vetor tem 5 posições
10
[ 1, 10, 15, 23, 8 ]
0 valor não foi encontrado
Info: End process (21:06:44)
```

Fonte: Compilação da autora.

▼ Aula 16) Funções

Toda função tem uma chamada, um conjunto de parâmetros (nem sempre), uma ação e um retorno.

Funções são ações executadas assim que são chamadas ou em decorrência de algum evento.

Uma função pode receber parâmetros e retornar um resultado.

Estrutura de uma função em JS:

```
nomeFuncao (parâmetro){
    // o código fica aqui. É especificado o que deve ser retornado
    return resultado
}

ação (5) // chamada da função + passagem de parâmetro em parêntese
s
```

Exemplo de uma função com números:

```
// Crie uma função que verifique se um número é par ou ímpar

function parOuImpar(n) {
    if (n % 2 == 0) {
```

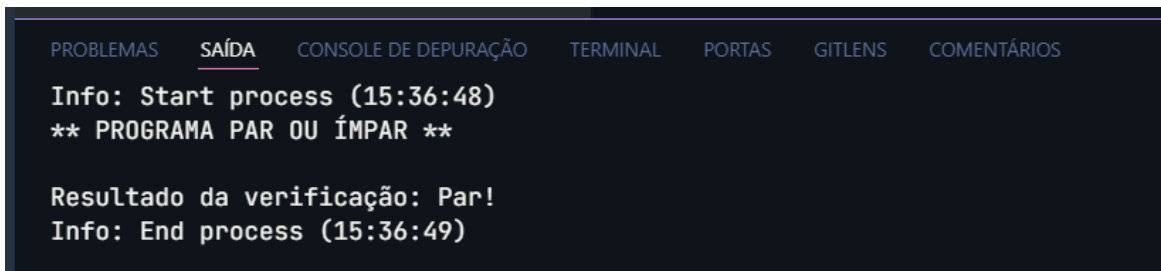
```

    return 'Par!'
  }
  else {
    return 'Ímpar!'
  }
}

console.log('** PROGRAMA PAR OU ÍMPAR **')
console.log('')
console.log('Resultado da verificação: ' + parOuImpar(2)) // chamada d
a função + apresentação do resultado

```

Resultado da execução:



```

PROBLEMAS  SAÍDA  CONSOLE DE DEPURANÇA  TERMINAL  PORTAS  GITLENS  COMENTÁRIOS
Info: Start process (15:36:48)
** PROGRAMA PAR OU ÍMPAR **

Resultado da verificação: Par!
Info: End process (15:36:49)

```

Fonte: Compilação da autora.

Exercício 02 de função:

```

// Crie uma função que calcule a soma de 2 números

function soma(n1, n2) {
  return n1 + n2
}

console.log('** PROGRAMA SOMA DE 2 NÚMEROS **')
console.log('')
console.log('Resultado da soma: ' + soma(2, 3)) // chamada da função
+ apresentação do resultado

```

Resultado da execução:

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS  GITLENS  COMENTÁRIOS

Info: Start process (15:36:48)
** PROGRAMA PAR OU ÍMPAR **

Resultado da verificação: Par!
Info: End process (15:36:49)
```

Fonte: Compilação da autora.

Exercício 03 de função:

```
// Crie uma função que calcule a soma de 2 números

function soma(n1 = 0, n2 = 0) { // n1 ou n2 valerão 0 se na chamada da f
  unção nenhum outro número for passado
    return n1 + n2
}

console.log('** PROGRAMA SOMA DE 2 NÚMEROS **')
console.log('')
console.log('Resultado da soma: ' + soma(7)) // chamada da função + a
  apresentação do resultado
```

Resultado da execução:

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS  GITLENS  COMENTÁRIOS

Info: Start process (15:41:18)
** PROGRAMA SOMA DE 2 NÚMEROS **

Resultado da soma: 7
Info: End process (15:41:18)
```

Fonte: Compilação da autora.

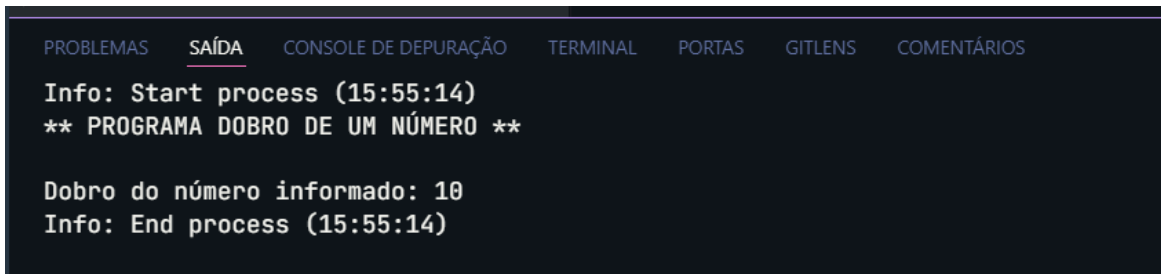
No JS é possível criar uma função DENTRO de uma variável. Exemplo exercício 04 de função:

```
// Criando uma função DENTRO de uma variável

let v = function(x) {
  return x * 2
}

console.log('** PROGRAMA DOBRO DE UM NÚMERO **')
console.log('')
console.log('Dobro do número informado: ' + v(5)) // chamada da função + apresentação do resultado
```

Resultado da execução:



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS  GITLENS  COMENTÁRIOS

Info: Start process (15:55:14)
** PROGRAMA DOBRO DE UM NÚMERO **

Dobro do número informado: 10
Info: End process (15:55:14)
```

Fonte: Compilação da autora.

Exercício com função recursiva (ou seja, uma função que chama ela própria):

```
// Crie uma função recursiva que calcule o fatorial de um número

/*

Lembrando que, fatorial é:
5! = 5 × 4 × 3 × 2 × 1 = 120

Com a função recursiva:
5! = 5 × 4!

*/
```



```
function fatorial(n){
  if (n == 1){
    return 1
  }
  else {
    return n * fatorial(n - 1) // é o mesmo que n x (n - 1)!
  }
}

console.log('** PROGRAMA FATORIAL DE UM NÚMERO USANDO FUNÇÃO RECURSIVA **')
console.log('')
console.log('Resultado do fatorial: ' + fatorial(5)) // chamada da função
+ apresentação do resultado
```

Resultado da execução:

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS  GITLENS  COMENTÁRIOS

Info: Start process (16:04:37)
** PROGRAMA FATORIAL DE UM NÚMERO USANDO FUNÇÃO RECURSIVA **

Resultado do fatorial: 120
Info: End process (16:04:37)
```

Fonte: Compilação da autora.

▼ Exercícios JavaScript (parte 7 e 8)

Código HTML:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Analisador de Números</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
```

```

<!-- Cabeçalho →
<header>
  <h1>Analisar de Números</h1>
</header>
<!-- Seção →
<section>
  <!-- div 1 →
  <div>
    Número (entre 1 e 100):
    <input type="number" name="fnum" id="fnum">
    <input type="button" value="Adicionar" onclick="adicionar()">
  <!-- chama a função adicionar() ao clicar sobre o botão →
    <br><br>
    <select name="flista" id="flista" size="6"></select>
    <br>
    <input type="button" value="Finalizar" onclick="finalizar()"> <!--
- chama a função finalizar() ao clicar sobre o botão →
  </div>
  <!-- div 2 →
  <div id="res">

  </div>
</section>
<!-- Rodapé →
<footer>
  <p>&copy; GabriellaXavier</p> <!-- Símbolo de copyright→
</footer>
<script src="script.js"></script>
</body>
</html>

```

Código JS:

```

// variáveis
let num = document.querySelector('input#fnum')
let lista = document.querySelector('select#flista')
let res = document.querySelector('div#res')

```

```

let valores = []

// fverifica se é ou não um número
function isNumero(n) {
  if (Number(n) >= 1 && Number(n) <= 100) { // se for um número entre
1 e 100
    return true
  }
  else { // senão...
    return false
  }
}

// verificando se o número já está na lista
function inLista(n, l) {
  if (l.indexOf(Number(n)) != -1) { // se o número já estiver na lista
    return true
  }
  else { // senão...
    return false
  }
}

// adicionar elementos ao select
function adicionar() {
  // verificando se é um número e se está na lista
  if (isNumero(num.value) && !inLista(num.value, valores)) {
    // adicionando os valores ao vetor valores[]
    valores.push(Number(num.value))

    // adicionando os valores no select
    let item = document.createElement('option')
    item.text = `Valor ${num.value} adicionado.`
    lista.appendChild(item) // adiciona mais um item na lista
    res.innerHTML = '' // limpa a div res depois de adicionar um elemento
  }
}

```

```

else { // se não for um número ou se já estiver na lista
    window.alert('Valor inválido ou já encontrado na lista.')
}

// apagando as informações do campo de texto
num.value = ''
num.focus()
}

function finalizar() {
    // verifica se o vetor está vazio
    if (valores.length == 0) {
        window.alert('Adicione valores antes de finalizar!')
    }
    else {
        let tot = valores.length // total de elementos no vetor
        let maior = valores[0] // maior valor
        let menor = valores[0] // menor valor
        let soma = 0 // soma
        let media = 0 // média

        // verificando o maior e o menor valor e somando os valores
        for (let pos in valores) {
            soma += valores[pos]

            if (valores[pos] > maior) {
                maior = valores[pos]
            }
            if (valores[pos] < menor) {
                menor = valores[pos]
            }
        }

        // calculando a média
        media = soma / tot

        res.innerHTML = '' // limpa a div res
        res.innerHTML += `<p>Ao todo, temos ${tot} números cadastrado

```

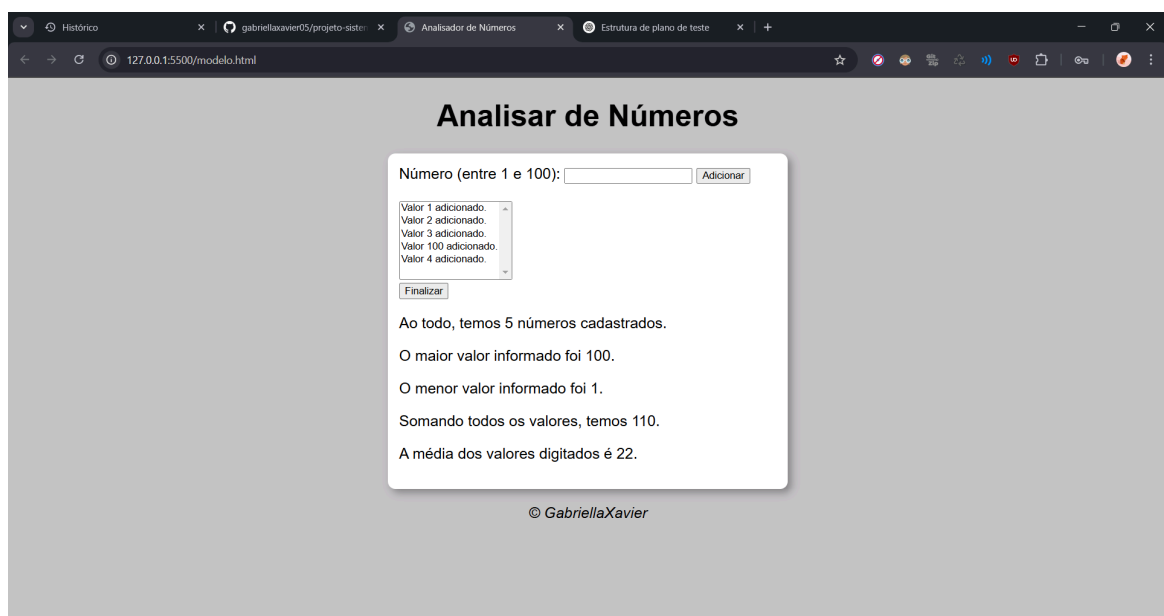
```

s.</p>`
    res.innerHTML += `<p>O maior valor informado foi ${maior}.</p>`
    res.innerHTML += `<p>O menor valor informado foi ${menor}.</p>`
    `

    res.innerHTML += `<p>Somando todos os valores, temos ${soma}.
</p>`
    res.innerHTML += `<p>A média dos valores digitados é ${media}.
</p>`
    }
}

```

Resultado:



Fonte: Compilação da autora.

▼ O que estudar daqui pra frente

Prosseguir nos estudos com os seguintes assuntos:

- HTML5 e CSS3;
- Functions;
- Objetos (JS também é uma linguagem orientada a objetos);
- Modularização [de códigos];
- RegEx (Expressões Regulares);

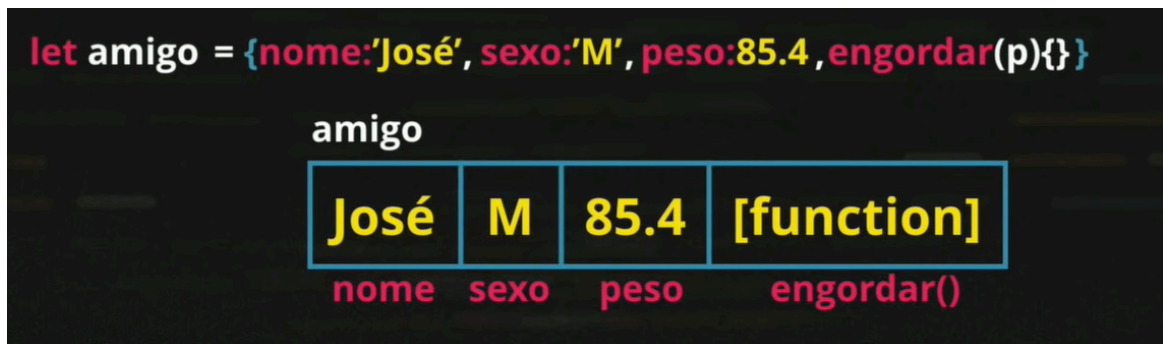
- JSON;
- AJAX;
- NodeJS.

Exemplo de POO em JavaScript

```
// declarando um array:
let num = [5, 8, 4]

// declarando um OBJETO em JS:
let amigo = {nome: 'José', sexo: 'M', peso: 85.4, engordar(p){}} // engor-
dar(p){} é uma função dentro do objeto 'amigo'
```

Representação da estrutura do objeto criado no código acima:



Fonte: Curso em Vídeo.

Objetos podem armazenar métodos (funcionalidades).

Obs.:

- Array é um objeto em JS;
- Objeto é um objto em JS. (coisa de doido, eu sei kkk 😂)

Modificando o código acima + resultado da execução:

```
objeto01.js U x
objeto01.js > ...
1  let amigo = {
2      nome: 'José',
3      sexo: 'M',
4      peso: 85.4,
5      engordar(p = 0){
6          console.log('Engordou')
7          this.peso += p // this é uma referência ao objeto
8      }
9  }
10
11  amigo.engordar(2) // passa o valor 2 para a função engordar
12
13  console.log(amigo)
14  console.log('')
15  console.log(amigo.nome + ' pesa ' + amigo.peso + ' kg.') // mostra somente o nome declarado no objeto

PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  PORTAS  GITLENS  COMENTÁRIOS  Filtro
Info: Start process (19:41:14)
Engordou
{ nome: 'José', sexo: 'M', peso: 87.4, engordar: [Function: engordar] }

José pesa 87.4 kg.
Info: End process (19:41:14)
```

Fonte: Compilação da autora.

Outras informações sobre JS

▼ **const**

É um tipo de variável que, uma vez atribuído um valor a ela, não dá para alterar. É como se fosse fixo.

▼ **let**

- **Escopo de Bloco:** A variável existe apenas no bloco onde foi declarada (entre `{ }`).

```
if (true) {
  let nome = "João";
  console.log(nome); // "João"
}
console.log(nome); // Erro: nome is not defined
```

- **Reatribuição permitida:** Você pode atribuir novos valores à variável.

```
let idade = 25;  
idade = 30;  
console.log(idade); // 30
```

- **Não permite declarações duplicadas** no mesmo escopo.

```
let cidade = "São Paulo";  
let cidade = "Rio de Janeiro"; // Erro: Identifier 'cidade' has already  
been declared
```

- **Temporal Dead Zone (TDZ):** A variável não pode ser acessada antes de ser declarada no código.

```
console.log(x); // Erro: Cannot access 'x' before initialization  
let x = 10;
```

▼ Diferenças `var` x `let`

A principal diferença entre `let` e `var` em JavaScript é o **escopo** e o **comportamento** de declaração.

1. Escopo:

- `let` tem **escopo de bloco**, ou seja, a variável declarada com `let` é acessível apenas dentro do bloco `{ }` onde foi definida (como em um `if`, `for` ou função).
- `var` tem **escopo de função**, o que significa que a variável declarada com `var` é acessível em toda a função, mesmo que declarada dentro de um bloco (como um `if`).

2. Declaração duplicada:

- `let` não permite que a mesma variável seja declarada mais de uma vez no mesmo escopo.

- `var` permite declarar a mesma variável várias vezes no mesmo escopo, o que pode causar confusão ou erros.

3. Hoisting:

- `let` é "hoisted" (elevação) para o topo do seu bloco, mas não pode ser acessado até a linha onde foi declarado, resultando na "**zona morta temporal**".
- `var` é "hoisted" para o topo da função ou escopo global e pode ser acessado antes da declaração, mas com valor `undefined`.

Resumo:

- `let` é mais seguro e previsível, com escopo de bloco e restrição a reatribuições no mesmo escopo.
- `var` é mais antigo, tem escopo de função e permite declarações duplicadas no mesmo escopo.