

Aula 7

Cores e acessibilidade

► Unidade

**Acessibilidade na web:
melhorando a experiência
do usuário**

O que vamos aprender?

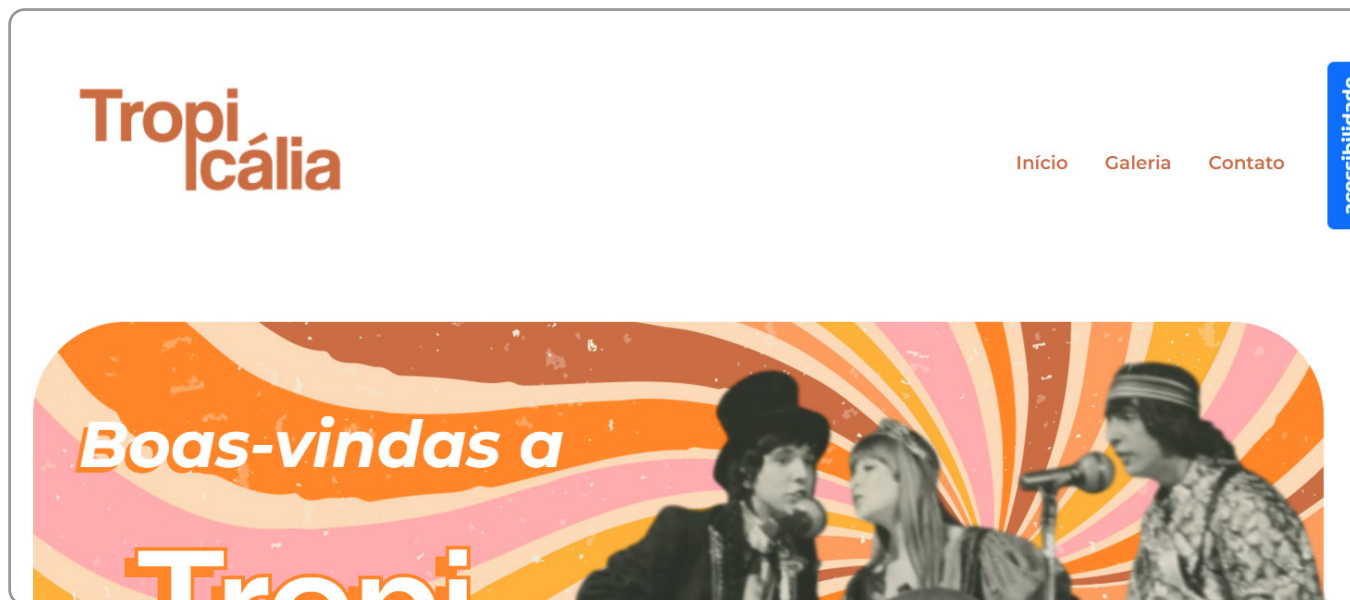
- Compreender a função dos atributos ARIA (Accessible Rich Internet Applications) na melhoria da acessibilidade em páginas web.
- Implementar atributos ARIA, como **aria-label** e **aria-expanded**, para melhorar a usabilidade de leitores de tela em elementos interativos.
- Manipular atributos ARIA via JavaScript para refletir mudanças dinâmicas na interface do usuário.

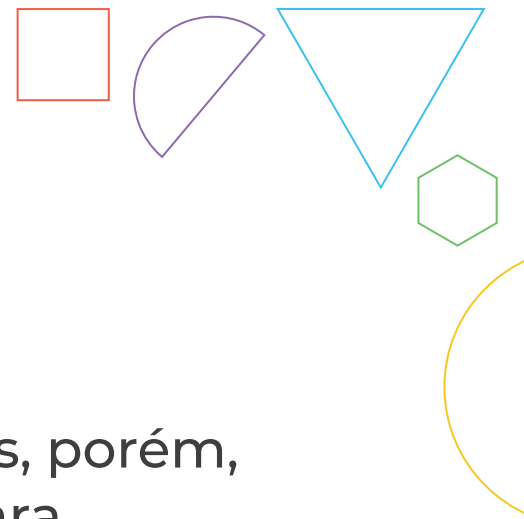


ACOMPANHE O VÍDEO DA AULA

Alterando o contraste

Na aula anterior, adicionamos mais um recurso de acessibilidade ao nosso site: o alto-contraste. Fizemos isso por meio da inclusão de um botão e de alterações no CSS. Agora, deixaremos nosso site ainda mais acessível para pessoas que fazem uso do recurso de leitor de tela utilizando os atributos ARIA.





Já sabemos como deixar as imagens do nosso site acessíveis, porém, existem outros símbolos que podem não ser perceptíveis para pessoas que usam o leitor de tela. Por exemplo, os botões do menu de acessibilidade, A+ e A-, e o botão de alto-contraste, no qual utilizamos um ícone.

Nesses casos, o leitor de tela lerá apenas o conteúdo do botão, e isso pode não ser percebido por quem usa um leitor de tela, pois o botão de alto-contraste, por exemplo, não possui texto para ser lido. Nesse caso, como faremos para que as pessoas que utilizam um leitor de tela saibam a função desse botão? É isso que exploraremos nesta aula.



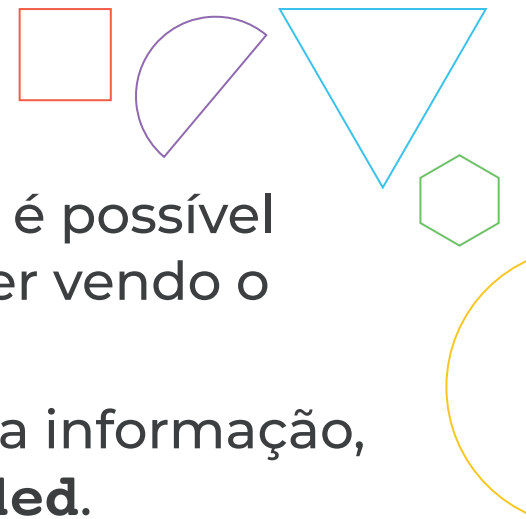
Para resolver esse problema, utilizaremos os atributos ARIA. Esses atributos servem para tornar os elementos interativos do HTML mais acessíveis para quem usa um leitor de tela. Ao adicionarmos o atributo **aria-label**, por exemplo, estamos fazendo com que o leitor de tela ignore o conteúdo de texto do botão e leia a informação que está nesse atributo.

Assim, no arquivo *index.html*, no VS Code, alteraremos o código dos botões do menu de acessibilidade. Adicionaremos os seguintes atributos e valores: **aria-label="Aumentar o tamanho da fonte"** para o botão com **id="aumentar-fonte"**; e **aria-label="diminuir o tamanho da fonte"** para o botão com **id="diminuir-fonte"**.

O código deve ficar da seguinte forma:

```
<div id="opcoes-acessibilidade" class="opcoes-acessibilidade
apresenta-lista">
  <button id="aumentar-fonte" class="btn btn-primary fw-bold"
aria-label="aumentar o tamanho da fonte">A+</button>
  <button id="diminuir-fonte" class="btn btn-primary fw-bold"
aria-label="diminuir o tamanho da fonte">A-</button>
  <button id="alterna-contraste" class="btn btn-primary fw-bold">
< i class="bi bi-shadows"></i></button>
</div>
```

Ao salvarmos o projeto e abrirmos a página no navegador, veremos que nenhuma mudança é perceptível. Isso ocorre porque esse recurso serve para identificar a função do botão para quem não consegue vê-lo, portanto, ele não produzirá um resultado visível.




Agora, se observarmos o site no navegador, veremos que só é possível saber se o menu de acessibilidade está aberto se você estiver vendo o conteúdo do site.

Para melhorar a experiência de quem não consegue ver essa informação, adicionaremos, no botão do menu, o atributo **aria-expanded**.

Além disso, precisaremos adicionar uma lógica ao código JavaScript, garantindo que o estado do menu (aberto ou fechado) seja apresentado corretamente para o leitor de tela.

Desse modo, no arquivo *index.html*, adicionaremos **aria-expanded="false"** no botão com **id="botao-acessibilidade"**. O primeiro estado é **false** porque o botão, inicialmente, está fechado, e será aberto quando clicarmos nele. O código ficará da seguinte forma:

```
<button id="botao-acessibilidade" class="btn btn-primary  
fw-bold -botao" aria-expanded="false" >acessibilidade</button>
```

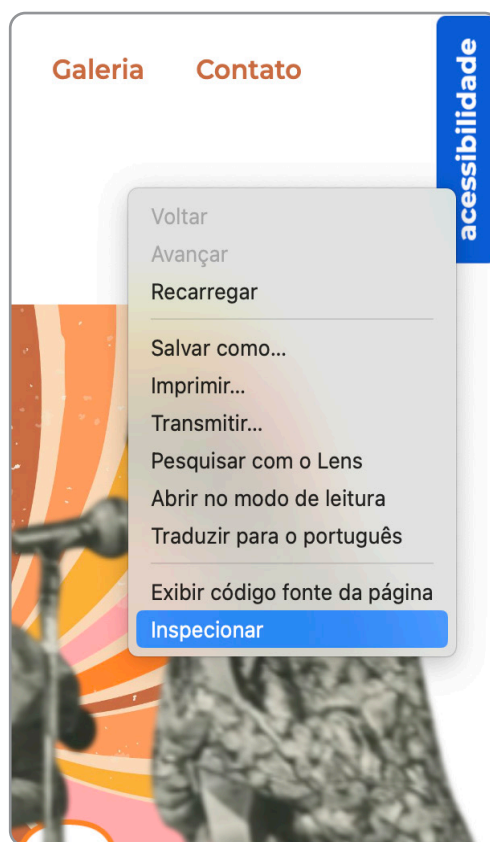


Agora, no arquivo *script.js*, adicionaremos a lógica de que, quando clicarmos no botão de acessibilidade, o valor do **aria-expanded** seja alterado para **true**. Dessa forma, no código JavaScript, já temos a ação de **click** no botão; então, vamos apenas acrescentar as seguintes linhas de código:

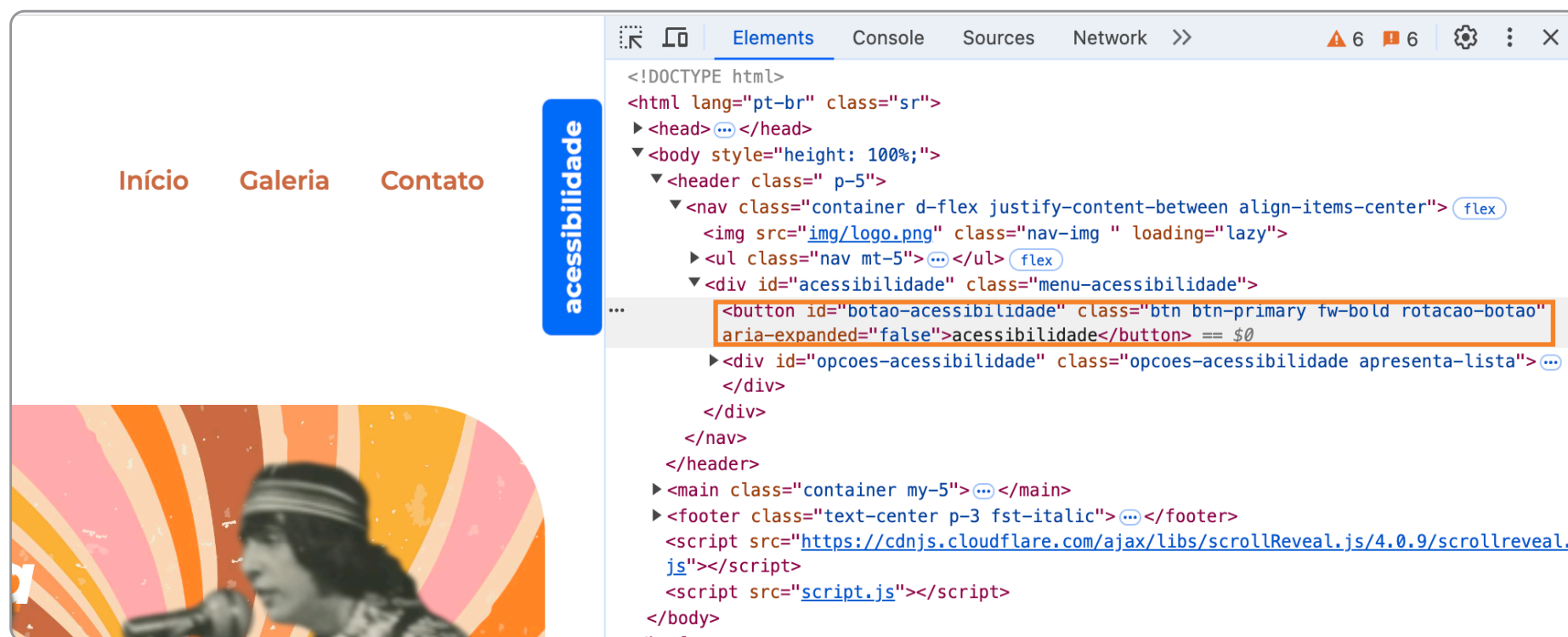
```
botaoDeAcessibilidade.addEventListener('click', function () {  
    botaoDeAcessibilidade.classList.toggle('rotacao-botao');  
    opcoesDeAcessibilidade.classList.toggle('apresenta-lista');  
  
    const botaoSelecionado = botaoDeAcessibilidade.getAttribute  
('aria-expanded') === 'true';  
    botaoDeAcessibilidade.setAttribute('aria-expanded',  
!botaoSelecionado);  
})
```

O que fizemos aqui foi pegar o valor atual do atributo **aria-expanded** e, então, verificar se ele é igual a **true**. Em seguida, alteramos o valor do atributo usando a função **setAttribute**, informando que o novo valor é o oposto do anterior. Sabendo que o oposto de **true** é **false**, então, sempre que clicarmos sobre o botão do menu, o atributo **aria-expanded** será chaveado entre esses dois valores, como desejado.

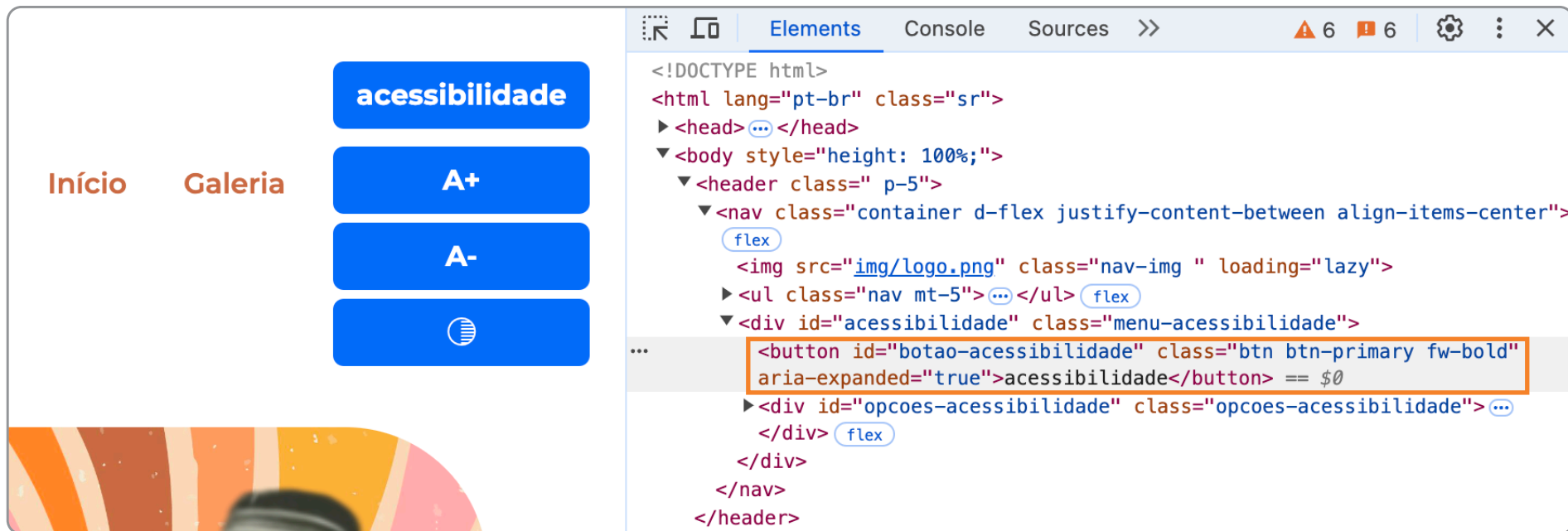
Para verificar se o que fizemos está funcionando corretamente, podemos abrir o site no navegador e inspecionar o código do botão. Para isso, clique sobre o menu de acessibilidade com o botão direito do mouse e selecione a opção *Inspecionar*, conforme a imagem a seguir:



No HTML, busque pelo elemento **button** e confira a diferença entre o botão de acessibilidade fechado e o que muda quando ele está aberto. Observe o código do menu de acessibilidade fechado:



Agora, observe o menu de acessibilidade aberto:



Assim, finalizamos mais uma aula! Agora, nosso site está ainda mais acessível para as pessoas que utilizam recursos de leitura de tela.

► Desafio

Nesta aula, aprendemos para que servem e como adicionar atributos ARIA em nosso site, de forma a deixá-lo mais acessível para quem usa leitores de tela. Vimos apenas dois tipos de atributos ARIA, mas existem outros. Seu desafio agora é pesquisar e identificar quais outros atributos ARIA existem e adicioná-los aos elementos interativos do seu site quando julgar necessário.



CLIQUE **AQUI** PARA AVALIAR ESTE MATERIAL