

# Componentes e Props

Fonte: Documentação oficial [React](#) e [Next JS](#)

# Conceitos básicos de React

Existem três conceitos principais do React com os quais você precisará estar familiarizado para começar a criar aplicativos React. Estes são:

- Components
- Props
- State

# Construindo IU com Componentes

As interfaces de usuário podem ser divididas em blocos de construção menores chamados componentes .

Os componentes permitem que você crie trechos de código autocontidos e reutilizáveis. Se você pensar em componentes como peças de LEGO , poderá pegar essas peças individuais e combiná-las para formar estruturas maiores. Se precisar atualizar uma parte da IU, você pode atualizar o componente ou bloco específico.

Media Component



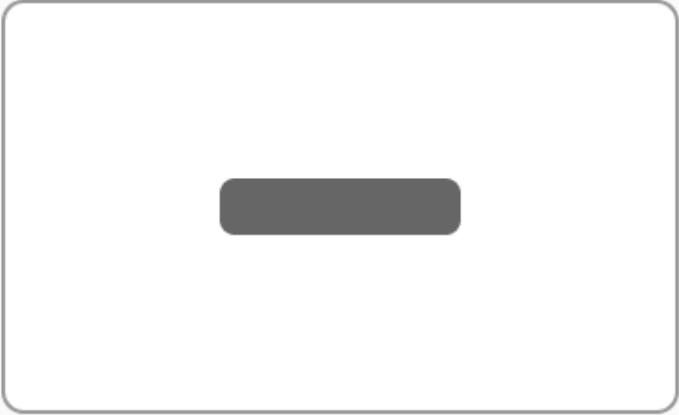
Image



Description



Button



Essa modularidade permite que seu código seja mais fácil de manter à medida que cresce porque você pode facilmente adicionar, atualizar e excluir componentes sem tocar no restante de nosso aplicativo.

O bom dos componentes React é que eles são apenas JavaScript. Vamos ver como você pode escrever um componente React, de uma perspectiva JavaScript:

# Criando componentes

No React, os componentes são funções. Dentro da sua tag script, escreva uma função chamada header:

```
<script type="text/jsx">
  const app = document.getElementById("app")

  function header() {
  }

  ReactDOM.render(<h1>Develop. Preview. Ship. 🚀 </h1>, app)
</script>
```

Um componente é uma função que retorna elementos da interface do usuário. Dentro da declaração de retorno da função, você pode escrever JSX:

```
<script type="text/jsx">
  const app = document.getElementById("app")

  function header() {
    return (<h1>Develop. Preview. Ship. 🚀 </h1>)
  }

  ReactDOM.render(, app)
</script>
```

Para renderizar este componente para o DOM, você pode passá-lo como primeiro argumento no método ReactDOM.render() :

```
<script type="text/jsx">

  const app = document.getElementById("app")

  function header() {
    return (<h1>Develop. Preview. Ship. 🚀 </h1>)
  }

  ReactDOM.render(header, app)
</script>
```



Mas, espere um segundo. Se você tentar executar o código acima em seu navegador, receberá um erro. Para que isso funcione, há duas coisas que você precisa fazer:

Primeiro, os componentes do React devem ser capitalizados para distingui-los do HTML simples e do JavaScript.

```
function Header() {  
  return <h1>Develop. Preview. Ship. 🚀 </h1>;  
}  
  
// Capitalize the React Component  
ReactDOM.render(Header, app);
```

Em segundo lugar, você usa componentes React da mesma forma que usaria tags HTML regulares, com colchetes angulares `<>`.

```
function Header() {  
  return <h1>Develop. Preview. Ship. 🚀 </h1>;  
}  
  
ReactDOM.render(<Header />, app);
```

# Componentes de aninhamento

Os aplicativos geralmente incluem mais conteúdo do que um único componente. Você pode aninhar os componentes do React uns dentro dos outros como faria com elementos HTML regulares.

No seu exemplo, crie um novo componente chamado HomePage:

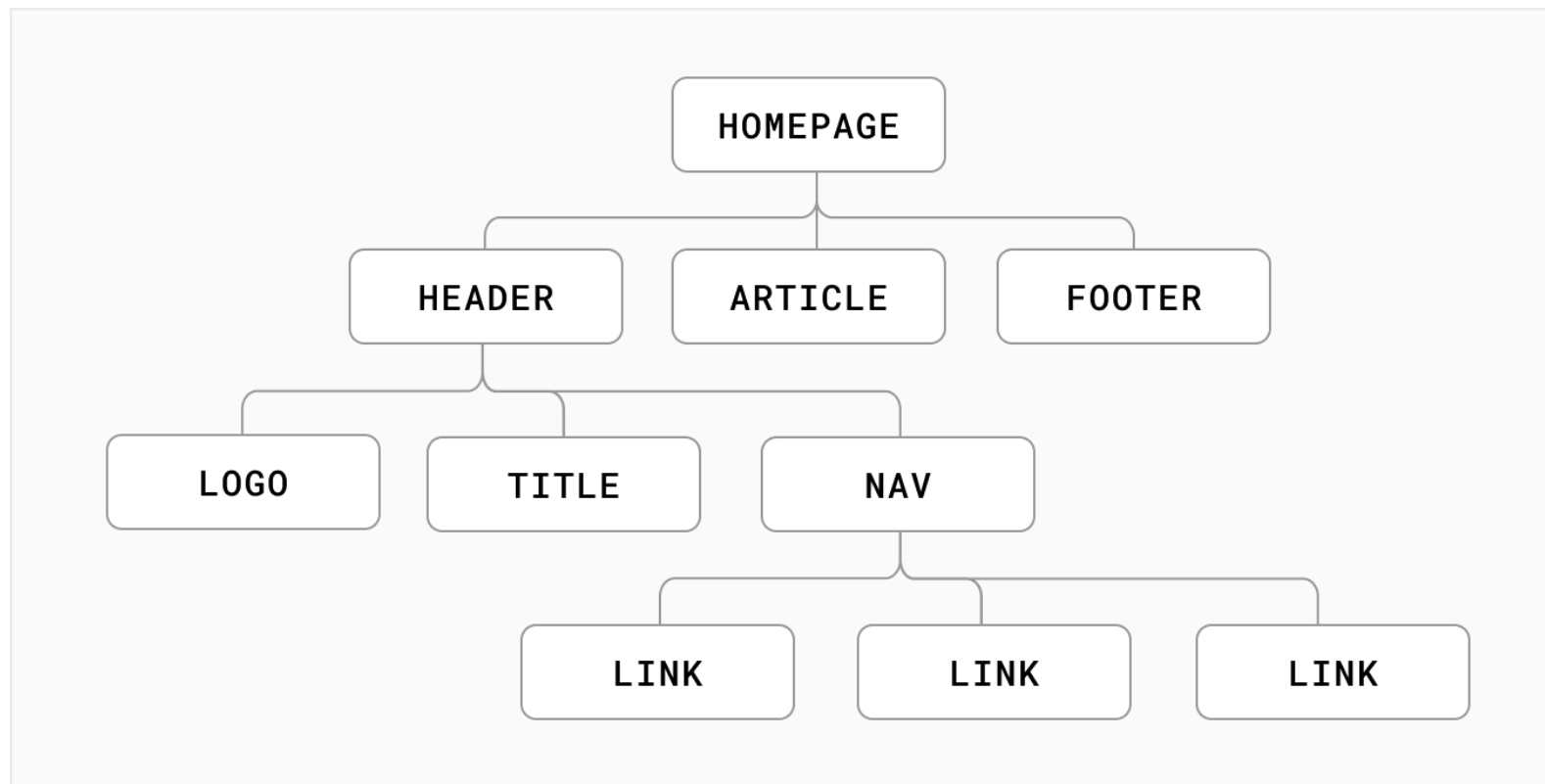
```
function Header() {  
  return <h1>Develop. Preview. Ship. 🚀 </h1>;  
}  
function HomePage() {  
  return <div></div>;  
}  
  
ReactDOM.render(<Header />, app);
```

Em seguida, aninhe o componente <Header> dentro do novo componente <HomePage>:

```
function Header() {  
  return <h1>Develop. Preview. Ship. 🚀 </h1>;  
}  
  
function HomePage() {  
  return (  
    <div>  
      { /* Nesting the Header component */ }  
      <Header />  
    </div>  
  );  
}  
  
ReactDOM.render(<Header />, app);
```

# Árvores de Componentes

Você pode continuar aninhando componentes React dessa maneira para formar árvores de componentes.



Por exemplo, seu componente de nível superior HomePage pode conter um Header, um Article e um Componente Footer. E cada um desses componentes poderia, por sua vez, ter seus próprios componentes filhos e assim por diante. Por exemplo, o componente Header pode conter um Logo, Title e um componente Navigation.

Este formato modular permite que você reutilize componentes em diferentes lugares dentro de seu aplicativo.

Em seu projeto, o `<HomePage>` é seu componente de nível superior, assim você pode passá-lo para o método `ReactDOM.render()` :

```
function Header() {  
  return <h1>Develop. Preview. Ship. 🚀</h1>;  
}  
  
function HomePage() {  
  return (  
    <div>  
      <Header />  
    </div>  
  );  
}  
  
ReactDOM.render(<HomePage />, app);
```

Como você aninharia um componente Header dentro de um componente Layout no React?

- a) `<Layout /><Cabeçalho />`
- b) `<Layout><Cabeçalho /></Layout>`
- c) `<layout><cabeçalho><cabeçalho/><layout/>`



Como você aninharia um componente Header dentro de um componente Layout no React?

- a) `<Layout /><Header />`
- b) `<Layout><Header /></Layout>`
- c) `<layout><header><header/><layout/>`

**Aprenda mais em:**

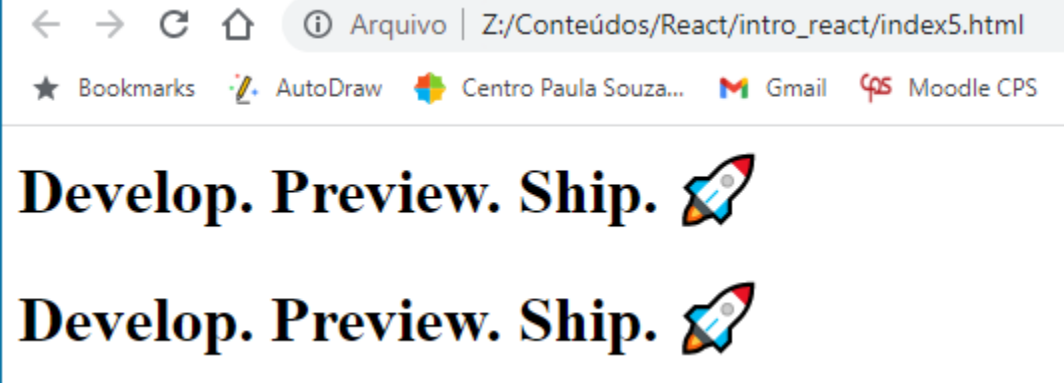
[Seu primeiro componente](#)

[Importando e Exportando Componentes](#)

# Exibindo Dados com Props

Até agora, se você reutilizasse seu componente `<Header />`, ele exibiria o mesmo conteúdo nas duas vezes.

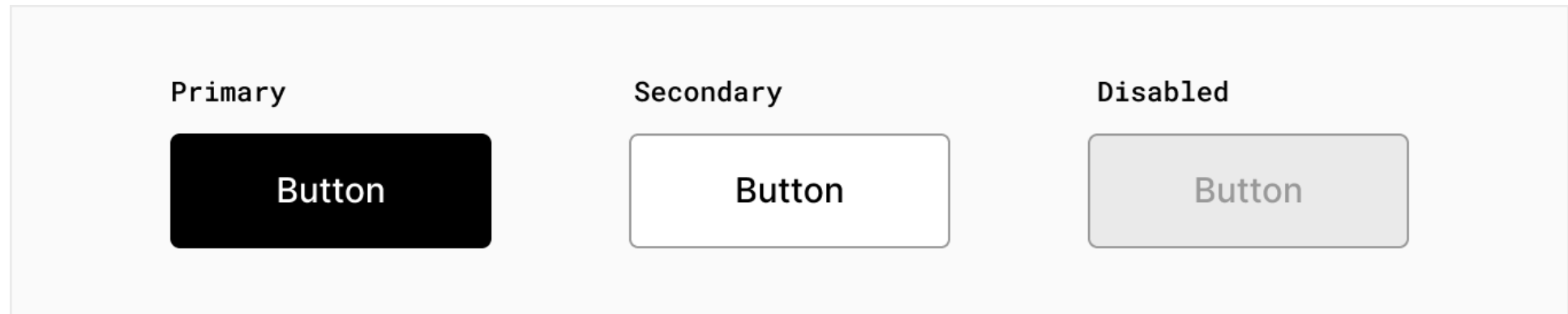
```
function Header() {  
  return <h1>Develop. Preview. Ship. 🚀</h1>;  
}  
  
function HomePage() {  
  return (  
    <div>  
      <Header />  
      <Header />  
    </div>  
  );  
}
```



Mas e se você quiser passar um texto diferente ou não souber as informações com antecedência porque está buscando dados de uma fonte externa?

Os elementos HTML regulares têm atributos que você pode usar para passar informações que alteram o comportamento desses elementos. Por exemplo, alterar o atributo `src` de um elemento `<img>` altera a imagem que é mostrada. Alterar o atributo `href` de uma tag `<a>` altera o destino do link.

Da mesma forma, você pode passar informações como propriedades para componentes React. Estes são chamados props.



Semelhante a uma função JavaScript, você pode criar componentes que aceitam argumentos personalizados (ou props) que alteram o comportamento do componente ou o que é mostrado visivelmente quando é renderizado na tela. Em seguida, você pode passar essas props de componentes pai para componentes filho.

**Nota:** *No React, os dados fluem pela árvore de componentes. Isso é conhecido como fluxo de dados unidirecional. O state, será discutido na posteriormente, pode ser passado de componentes pai para filho como props.*

# Usando props

Em seu componente HomePage, você pode passar uma propriedade title personalizada para o componente Header, assim como passaria atributos HTML:

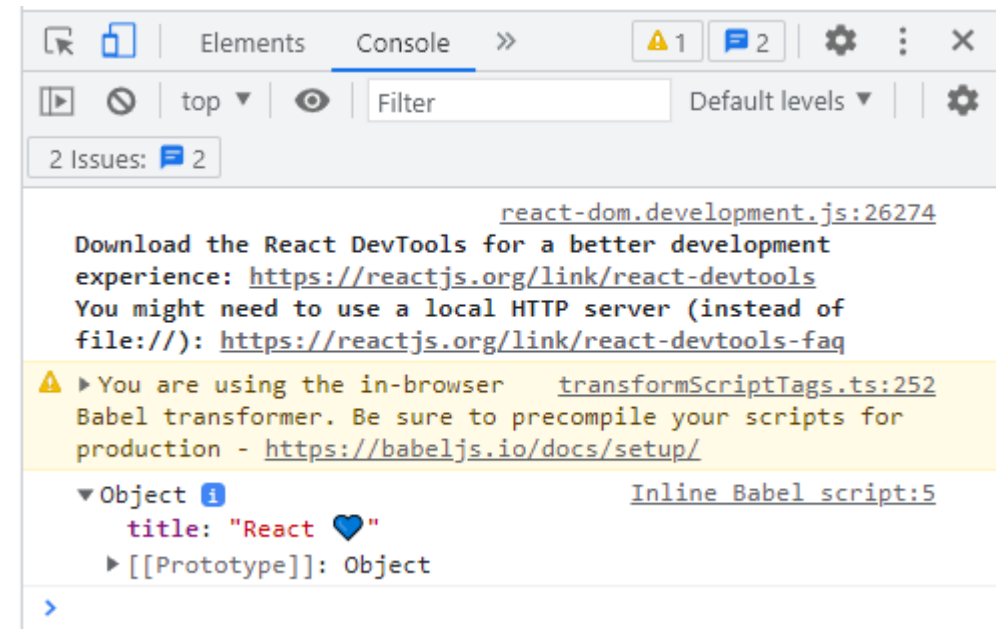
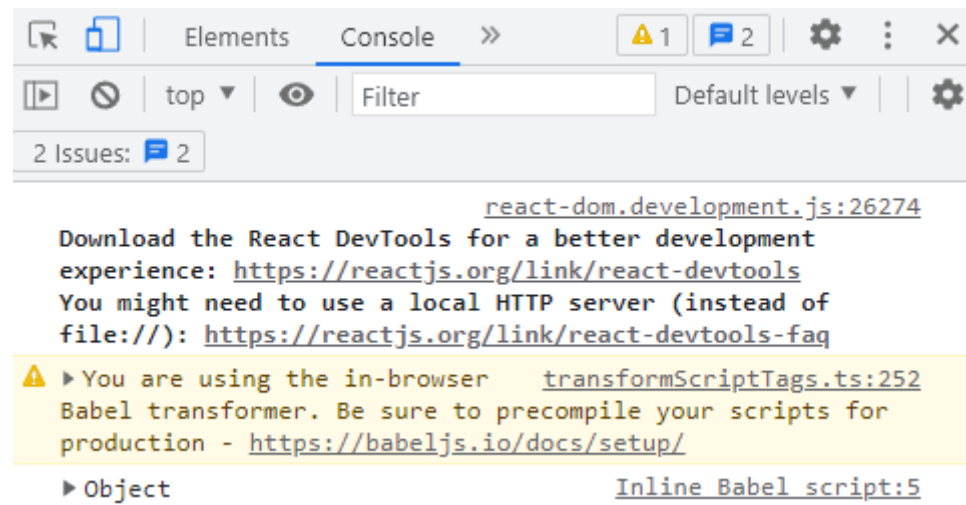
```
function HomePage() {  
  return (  
    <div>  
      <Header title="React ❤️" />  
    </div>  
  );  
}
```

E Header, o componente filho, pode aceitar essas props como seu primeiro parâmetro de função:

```
function Header(props) {  
  // return <h1>Develop. Preview. Ship. 🚀</h1>  
  // }  
  
  // function HomePage() {  
  //   return (  
  //     <div>  
  //       <Header title="React ❤️" />  
  //     </div>  
  //   )  
  // }
```

Se você der um `console.log()` de props, você pode ver que é um objeto com uma propriedade de título.

```
function Header(props) {  
  console.log(props)  
  return <h1>Develop. Preview. Ship. 🚀</h1>;  
}
```



Como props é um objeto, você pode usar a desestruturação de objeto para nomear explicitamente os valores de props dentro dos parâmetros de sua função:

```
function Header({title}) {  
  console.log(title)  
  return <h1>Develop. Preview. Ship. 🚀 </h1>;  
}
```

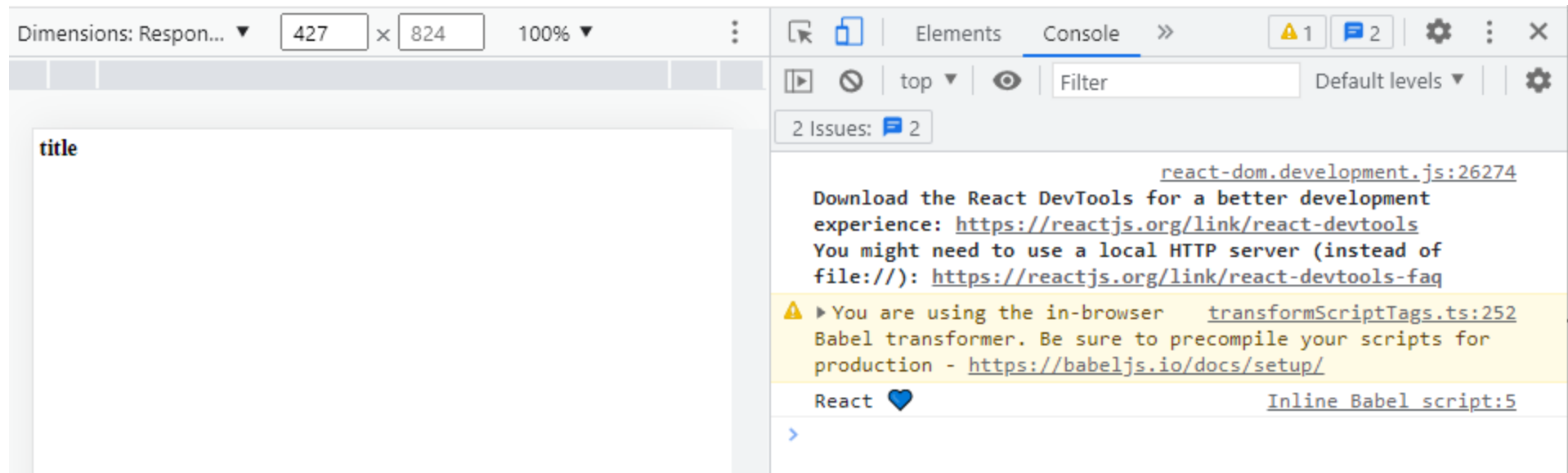


Em seguida, você pode substituir o conteúdo da tag `<h1>` por sua variável de título.

```
function Header({ title }) {  
  console.log(title);  
  return <h1>title</h1>;  
}
```

Se você abrir seu projeto no navegador, verá que ele está exibindo a palavra “title”. Isso ocorre porque o React acha que você pretende renderizar uma string de texto simples para o DOM.

Você precisa de uma maneira de indicar ao React que esta é uma variável JavaScript.



# Usando variáveis em JSX

Para usar a variável que você definiu, você pode usar chaves {}, uma sintaxe JSX especial que permite escrever JavaScript regular diretamente dentro de sua marcação JSX.

```
function Header({title}) {  
  return <h1>{title}</h1>;  
}
```

Você pode pensar em chaves como uma maneira de inserir código “na terra do JavaScript” enquanto estiver na “terra do JSX”. Você pode adicionar qualquer expressão JavaScript (algo que seja avaliado como um único valor) dentro de chaves.

Por exemplo:

1. Uma propriedade de objeto com notação de ponto.

```
function Header(props) {  
  |   return <h1>{props.title}</h1>;  
}
```

2. Um modelo literal :

```
function Header({ title }) {  
  |   return <h1>{`Cool ${title}`}</h1>;  
}
```

### 3. O valor retornado de uma função.

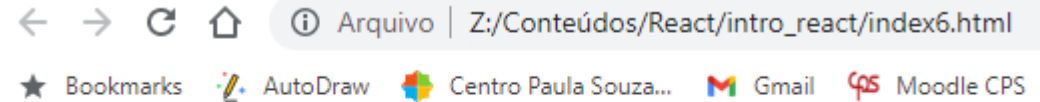
```
function createTitle(title) {  
  if (title) {  
    return title;  
  } else {  
    return 'Default title';  
  }  
}  
  
function Header({ title }) {  
  return <h1>{createTitle(title)}</h1>;  
}
```

### 4. Ou operadores ternários.

```
function Header({ title }) {  
  return <h1>{title ? title : 'Default Title'}</h1>;  
}
```

Agora você pode passar qualquer String para sua propriedade de título e, como você considerou o caso padrão em seu componente com o operador ternário, você pode até mesmo não passar uma propriedade de título:

```
function Header({ title }) {  
  return <h1>{title ? title : 'Default title'}</h1>;  
}  
  
function HomePage() {  
  return (  
    <div>  
      <Header />  
    </div>  
  );  
}
```



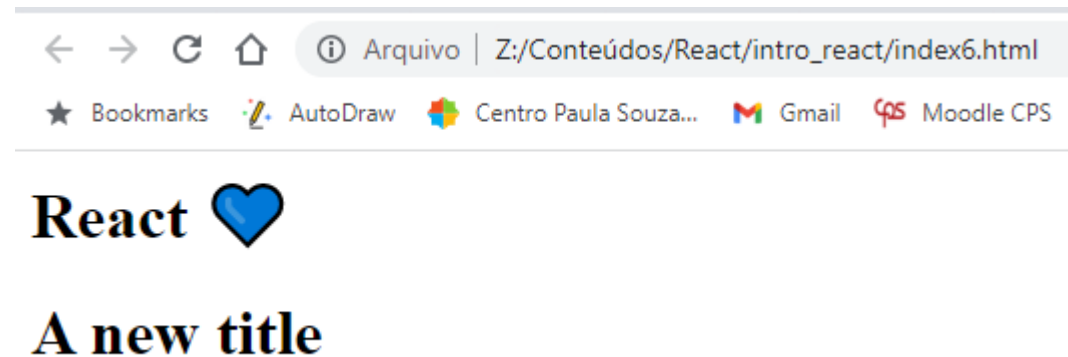
← → ↻ 🏠 ⓘ Arquivo | Z:/Conteúdos/React/intro\_react/index6.html

★ Bookmarks 🎨 AutoDraw 🌐 Centro Paula Souza... 📧 Gmail 📎 Moodle CPS

**Default title**

Seu componente agora aceita uma propriedade de título genérica que você pode reutilizar em diferentes partes de seu aplicativo. Tudo o que você precisa fazer é alterar o título:

```
function HomePage() {  
  return (  
    <div>  
      <Header title="React ❤️" />  
      <Header title="A new title" />  
    </div>  
  );  
}
```



# Iterando por listas

É comum ter dados que você precisa mostrar como uma lista. Você pode usar métodos de matriz para manipular seus dados e gerar elementos de interface do usuário com estilo idêntico, mas com informações diferentes.

**Observação:** *o React não tem opinião quando se trata de busca de dados, o que significa que você pode escolher a solução que melhor atende às suas necessidades. Por enquanto, você pode usar um array simples para representar os dados.*



# Adicione um array de nomes ao seu componente HomePage:

```
function HomePage() {  
  → const names = ['Ada Lovelace', 'Grace Hopper', 'Margaret Hamilton'];  
  
  return (  
    <div>  
      <Header title="React ❤️" />  
      <Header title="A new title" />  
    </div>  
  );  
}
```

Você pode usar o método `array.map()` para iterar sobre a matriz e usar uma arrow function para mapear um nome para um item da lista:

```
function HomePage() {  
  
  const names = ['Ada Lovelace', 'Grace Hopper', 'Margaret Hamilton'];  
  
  return (  
    <div>  
      <Header title="Develop. Preview. Ship. 🚀" />  
      <ul>  
        {names.map((name) => (  
          <li>{name}</li>  
        ))}  
      </ul>  
    </div>  
  );  
}
```


Observe como você usou chaves para entrar e sair da terra "JavaScript" e "JSX".

Se você executar este código, o React nos dará um aviso sobre uma prop "key" ausente. Isso ocorre porque o React precisa de algo para identificar exclusivamente os itens em uma matriz para saber quais elementos atualizar no DOM.

```
✖ ▶ Warning: react.development.js:245  
Each child in a list should have a  
unique "key" prop.  
  
Check the render method of `HomePage`.  
See https://reactjs.org/link/warning-keys  
for more information.  
    at li  
    at HomePage
```

Você pode usar os nomes por enquanto, já que eles são exclusivos, mas é recomendável usar algo garantido como exclusivo, como um ID de item.

```
<ul>
  {names.map((name) => (
    <li key={name}>{name}</li>
  ))}
</ul>
```



# Para que servem os props no React?

- a) Escrever regras de CSS
- b) Passar informações para componentes
- c) Adicionar atributos a elementos HTML

# Para que servem os props no React?

- a) Escrever regras de CSS
- b) **Passar informações para componentes**
- c) Adicionar atributos a elementos HTML

## **Aprenda mais em:**

[Passando props para um componente](#)

[Listas de renderização](#)

[Renderização Condicional](#)