

Overview

Components: UI building blocks

Defining a component

Step 1: Export the component

Step 2: Define the function

Step 3: Add markup

Using a component

What the browser sees

Nesting and organizing components

Recap

Challenges

Your First Component

Components are one of the core concepts of React. They are the foundation upon which you build user interfaces (UI), which makes them the perfect place to start your React journey!

You will learn

- What a component is
- What role components play in a React application
- How to write your first React component

Components: UI building blocks

On the Web, HTML lets us create rich structured documents with its built-in set of tags like `<h1>` and ``:

```
<article>
  <h1>My First Component</h1>
  <ol>
    <li>Components: UI Building Blocks</li>
    <li>Defining a Component</li>
    <li>Using a Component</li>
  </ol>
</article>
```

This markup represents this article `<article>`, its heading `<h1>`, and an (abbreviated) table of contents as an ordered list ``. Markup like this, combined with CSS for style, and JavaScript for interactivity, lies behind

every sidebar, avatar, modal, dropdown—every piece of UI you see on the Web.

React lets you combine your markup, CSS, and JavaScript into custom “components”, **reusable UI elements for your app**. The table of contents code you saw above could be turned into a `<TableOfContents />` component you could render on every page. Under the hood, it still uses the same HTML tags like `<article>`, `<h1>`, etc.

Just like with HTML tags, you can compose, order and nest components to design whole pages. For example, the documentation page you’re reading is made out of React components:

```
<PageLayout>
  <NavigationHeader>
    <SearchBar />
    <Link to="/docs">Docs</Link>
  </NavigationHeader>
  <Sidebar />
  <PageContent>
    <TableOfContents />
    <DocumentationText />
  </PageContent>
</PageLayout>
```

As your project grows, you will notice that many of your designs can be composed by reusing components you already wrote, speeding up your development. Our table of contents above could be added to any screen with `<TableOfContents />` ! You can even jumpstart your project with the thousands of components shared by the React open source community like [Chakra UI](#) and [Material UI](#).

Defining a component

Traditionally when creating web pages, web developers marked up their content and then added interaction by sprinkling on some JavaScript. This worked great when interaction was a nice-to-have on the web. Now it is

sprinkling on some JavaScript. This worked great when interaction was a nice-to-have on the web. Now it is expected for many sites and all apps. React puts interactivity first while still using the same technology: **a React component is a JavaScript function that you can *sprinkle with markup***. Here's what that looks like (you can edit the example below):

App.js

Download Reset Fork

```
1 export default function Profile() {
2   return (
3     
7   )
8 }
9
```



And here's how to build a component:

Step 1: Export the component

The `export default` prefix is a [standard JavaScript syntax](#) (not specific to React). It lets you mark the main function in a file so that you can later import it from other files. (More on importing in [Importing and Exporting Components!](#))

Step 2: Define the function

With `function Profile() { }` you define a JavaScript function with the name `Profile`.

Pitfall

React components are regular JavaScript functions, but **their names must start with a capital letter** or they won't work!

Step 3: Add markup

The component returns an `` tag with `src` and `alt` attributes. `` is written like HTML, but it is actually JavaScript under the hood! This syntax is called **JSX**, and it lets you embed markup inside JavaScript.

Return statements can be written all on one line, as in this component:

```
return ;
```

But if your markup isn't all on the same line as the `return` keyword, you must wrap it in a pair of parentheses:

```
return (  
  <div>  
      
  </div>  
);
```

Pitfall

Without parentheses, any code on the lines after `return` **will be ignored!**

Using a component

Now that you've defined your `Profile` component, you can nest it inside other components. For example, you can export a `Gallery` component that uses multiple `Profile` components:

App.js

Download Reset Fork

```
1 function Profile() {  
2   return (  
3       
7   );  
8 }  
9  
10 export default function Gallery() {  
11   return (  
12     <section>  
13       <h1>Amazing scientists</h1>  
14       <Profile />  
15       <Profile />  
16       <Profile />  
17     </section>  
18   );  
19 }
```

Amazing scientists



▼ Show more

What the browser sees

Notice the difference in casing:

- `<section>` is lowercase, so React knows we refer to an HTML tag.
- `<Profile />` starts with a capital `P`, so React knows that we want to use our component called `Profile`.

And `Profile` contains even more HTML: ``. In the end, this is what the browser sees:

```
<section>
  <h1>Amazing scientists</h1>
  
  
  
</section>
```

Nesting and organizing components

Components are regular JavaScript functions, so you can keep multiple components in the same file. This is convenient when components are relatively small or tightly related to each other. If this file gets crowded, you can always move `Profile` to a separate file. You will learn how to do this shortly on the [page about imports](#).


Because the `Profile` components are rendered inside `Gallery`—even several times!—we can say that `Gallery` is a **parent component**, rendering each `Profile` as a “child”. This is part of the magic of React: you can define a component once, and then use it in as many places and as many times as you like.

Pitfall

Components can render other components, but **you must never nest their definitions**:

```
export default function Gallery() {
  // 🚫 Never define a component inside another component!
  function Profile() {
    // ...
  }
  // ...
}
```

The snippet above is [very slow and causes bugs](#). Instead, define every component at the top level:

```
export default function Gallery() {  
  // ...  
}  
  
//  Declare components at the top level  
function Profile() {  
  // ...  
}
```

When a child component needs some data from a parent, [pass it by props](#) instead of nesting definitions.

DEEP DIVE

Components all the way down

 Hide Details

Your React application begins at a “root” component. Usually, it is created automatically when you start a new project. For example, if you use [CodeSandbox](#) or if you use the framework [Next.js](#), the root component is defined in `pages/index.js`. In these examples, you’ve been exporting root components.

Most apps use components all the way down. This means that you won’t only use components

Most React apps use components all the way down. This means that you won't only use components for reusable pieces like buttons, but also for larger pieces like sidebars, lists, and ultimately, complete pages! Components are a handy way to organize UI code and markup, even if some of them are only used once.

[React-based frameworks](#) take this a step further. Instead of using an empty HTML file and letting React “take over” managing the page with JavaScript, they *also* generate the HTML automatically from your React components. This allows your app to show some content before the JavaScript code loads.

Still, many websites only use React to [add interactivity to existing HTML pages](#). They have many root components instead of a single one for the entire page. You can use as much—or as little—React as you need.

Recap

You've just gotten your first taste of React! Let's recap some key points.

- React lets you create components, **reusable UI elements for your app**.
- In a React app, every piece of UI is a component.
- React components are regular JavaScript functions except:
 1. Their names always begin with a capital letter.
 2. They return JSX markup.

Try out some challenges

Try out some challenges

1. Export the component 2. Fix the return statement 3. Spot the mistake 4. Your own component



Challenge 1 of 4: Export the component

This sandbox doesn't work because the root component is not exported:

App.js

Download Reset Fork

```
1 function Profile() {  
2   return (  
3       
7   );  
8 }  
9
```

Unable to establish connection with the sandpack bundler. Make sure you are online or try again later. If the problem persists, please report it via email or submit an issue on GitHub.

Try to fix it yourself before looking at the solution!

Show solution

Next Challenge >

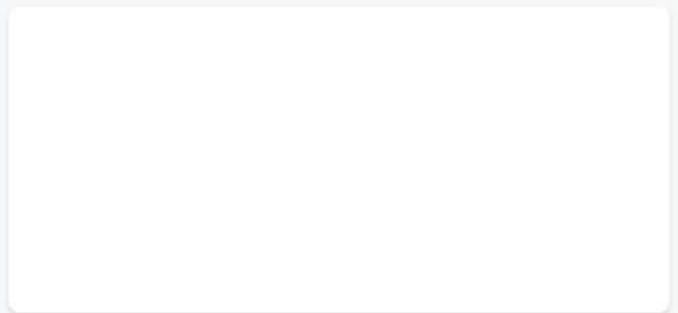
Solution

Add `export default` before the function definition like so:

App.js

Download Reset Fork

```
1 export default function Profile() {  
2   return (  
3       
7   );  
8 }  
9
```



You might be wondering why writing `export` alone is not enough to fix this example. You can learn the difference between `export` and `export default` in [Importing and Exporting Components](#).

Close solution

Next Challenge >

Try out some challenges

1. Export the component 2. Fix the return statement 3. Spot the mistake 4. Your own component



Challenge 2 of 4: Fix the return statement

Something isn't right about this `return` statement. Can you fix it?

App.js

Download Reset Fork

```
1 export default function Profile() {  
2   return  
3     

 Hide hint

 Show solution

Next Challenge >

You may get an “Unexpected token” error while trying to fix this. In that case, check that the semicolon appears *after* the closing parenthesis. Leaving a semicolon inside `return ( )` will cause an error.

## Solution

You can fix this component by moving the return statement to one line like so:

App.js Download Reset Fork

```
1 export default function Profile() {
2 return
3 }
4
```



Or by wrapping the returned JSX markup in parentheses that open right after `return`:

App.js Download Reset Fork

```
1 export default function Profile() {
2 return (
3
7);
8 }
9
```

Unable to establish connection with the sandpack bundler. Make sure you are online or try again later. If the problem persists, please report it via email or submit an issue on GitHub.

### Challenge 3 of 4: Spot the mistake

Something's wrong with how the `Profile` component is declared and used. Can you spot the mistake? (Try to remember how React distinguishes components from the regular HTML tags!)

App.js

[Download](#) [Reset](#) [Fork](#)

```
1 function profile() {
2 return (
3
7);
8 }
9
10 export default function Gallery() {
11 return (
12 <section>
13 <h1>Amazing scientists</h1>
14 <profile />
15 <profile />
16 <profile />
17 </section>
18);
19 }
20
```



^ Show less

## Solution

React component names must start with a capital letter.

Change `function profile()` to `function Profile()`, and then change every `<profile />` to `<Profile />`:

App.js

Download Reset Fork

```
1 function Profile() {
2 return (
3
7);
8 }
9
10 export default function Gallery() {
11 return (
12 <section>
13 <h1>Amazing scientists</h1>
14 <Profile />
15 <Profile />
16 <Profile />
17 </section>
18);
19 }
```

▼ Show more

### Amazing scientists



Close solution

Next Challenge >



## Try out some challenges

1. Export the component 2. Fix the return statement 3. Spot the mistake 4. Your own component



### Challenge 4 of 4: Your own component

Write a component from scratch. You can give it any valid name and return any markup. If you're out of ideas, you can write a `Congratulations` component that shows `<h1>Good job!</h1>`. Don't forget to export it!

App.js

 Download  Reset  Fork

```
1 // Write your component below!
2
3
```

 Show solution

## Solution

App.js

[Download](#) [Reset](#) [Fork](#)

```
1 export default function Congratulations() {
2 return (
3 <h1>Good job!</h1>
4);
5 }
6
```

Good job!

Close solution