3 Rendering Elements

These docs are old and won't be updated. Go to react.dev for the new React docs.

These new documentation pages teach how to write JSX and show it on an HTML page:

- Writing Markup with JSX
- Add React to an Existing Project

Elements are the smallest building blocks of React apps.

An element describes what you want to see on the screen:

```
const element = <h1>Hello, world</h1>;
```

Unlike browser DOM elements, React elements are plain objects, and are cheap to create. React DOM takes care of updating the DOM to match the React elements.

Note:

One might confuse elements with a more widely known concept of "components". We will introduce components in the next section. Elements are what components are "made of", and we encourage you to read this section before jumping ahead.

Rendering an Element into the DOM

Let's say there is a `<div>` somewhere in your HTML file:

```
<div id="root"></div>
```

We call this a "root" DOM node because everything inside it will be managed by React DOM.

Applications built with just React usually have a single root DOM node. If you are integrating React into an existing app, you may have as many isolated root DOM nodes as you like.

To render a React element, first pass the DOM element to `ReactDOM.createRoot()`, then pass the React element to `root.render()`:

```
const root = ReactDOM.createRoot(
  document.getElementById('root')
);
const element = <h1>Hello, world</h1>;
root.render(element);
```

Try it on CodePen

It displays “Hello, world” on the page.

Updating the Rendered Element

React elements are immutable. Once you create an element, you can’t change its children or attributes. An element is like a single frame in a movie: it represents the UI at a certain point in time.
With our knowledge so far, the only way to update the UI is to create a new element, and pass it to `root.render()`.

Consider this ticking clock example:

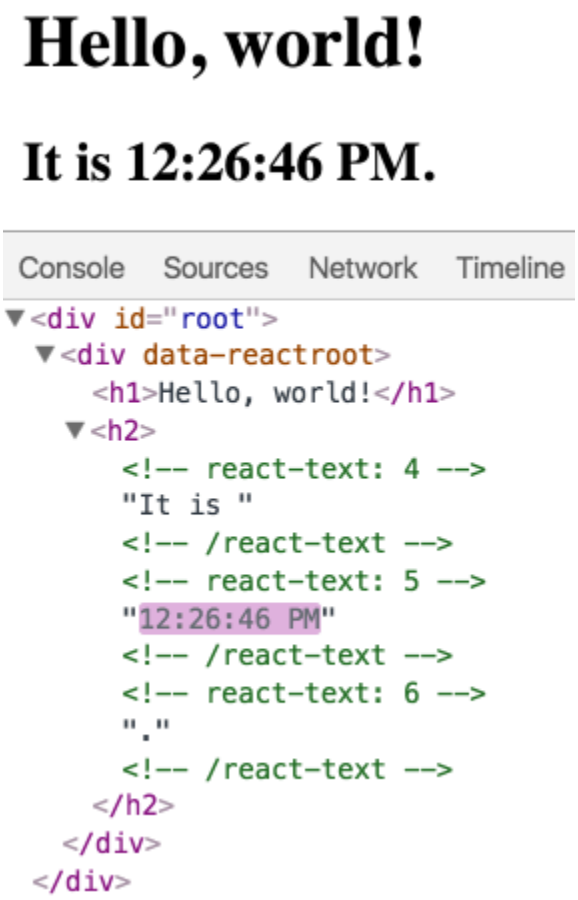
```
const root = ReactDOM.createRoot(
  document.getElementById('root')
);

function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  root.render(element);}

setInterval(tick, 1000);
```

Try it on CodePen
It calls `root.render()` every second from a `setInterval()` callback.
Note:
In practice, most React apps only call `root.render()` once. In the next sections we will learn how such code gets encapsulated into stateful components.

We recommend that you don’t skip topics because they build on each other.

React Only Updates What's Necessary

React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.
You can verify by inspecting the last example with the browser tools:



Even though we create an element describing the whole UI tree on every tick, only the text node whose contents have changed gets updated by React DOM.

In our experience, thinking about how the UI should look at any given moment, rather than how to change it over time, eliminates a whole class of bugs.