When you create a web page and then open it in Dev Tools (F12) in the elements tab you’ll see what looks like the source code. However it is not actually the source code; it is the DOM! And the difference is that the DOM is a combination of your original source code + changes that you made to it via JavaScript. So the HTML code you see, if it were modified by JavaScript, is *not the same* as the original HTML code that was sent to the browser.

Imperative programming is where you specify (in this case) the JavaScript which tells the browser, step by step, what to do so as to add an HTML element, for example.

But Declarative programming is where you simply declare something like, “I want to add an H1 tag with some text” and you don’t specify how to do it; you let the “system” figure that out for you. This is what React does. React is a “declarative library.”

Declarative programming allows you to describe what you want to happen, rather than the steps to make it happen.

In a React App we usually have a lot of JSX. Browsers don’t understand JSX, so we need to have a JavaScript compiler, such as “Babel” to transform our JSX code into regular JavaScript.

There are three core concepts of React that you'll need to be familiar with to start building React applications. These are:

* Components
* Props
* State

[**Creating components**](https://nextjs.org/learn/react-foundations/building-ui-with-components#creating-components)

In React, components are **functions.** Inside your script tag, create a new function called header:

A component is a function that **returns UI elements**. Inside the return statement of the function, you can write JSX:

First, React components should be capitalized to distinguish them from plain HTML and JavaScript:

you can pass pieces of information as properties to React components. These are called props.

Similar to a JavaScript function, you can design components that accept custom arguments (or props) that change the component's behavior or what is visibly shown when it's rendered to the screen. Then, you can pass down these props from parent components to child components

**Note:** In React, data flows down the component tree. This is referred to as one-way data flow. State, which will be discussed in the next chapter, can be passed from parent to child components as props.

You can think of curly braces as a way to enter "JavaScript land" while you are in "JSX land". You can add any **JavaScript expression** (something that evaluates to a single value) inside curly braces.

Pick up at:

[React Foundations: Displaying Data with Props | Next.js](https://nextjs.org/learn/react-foundations/displaying-data-with-props)

[React Foundations: Adding Interactivity with State | Next.js](https://nextjs.org/learn/react-foundations/updating-state)

the React hook used to manage state is called: useState()

Add useState() to your project. It returns an array, and you can access and use those array values inside your component using **array destructuring**:

[React Foundations: From React to Next.js | Next.js](https://nextjs.org/learn/react-foundations/from-react-to-nextjs)