What is a component and what does a component return in React?

A component is a reusable block of code that represents a user interface which is an element like a button, input field, forms, navigation bars ... A react component is similar to a JavaScript function or class that defines how a part of the UI should be rendered. There are 2 types of components. **Function components** and **class components**.

In React, a component always returns a React element or a tree of React elements. A React element is a plain JavaScript object that represents a virtual DOM element, describing what should be rendered on the screen. An element can be:

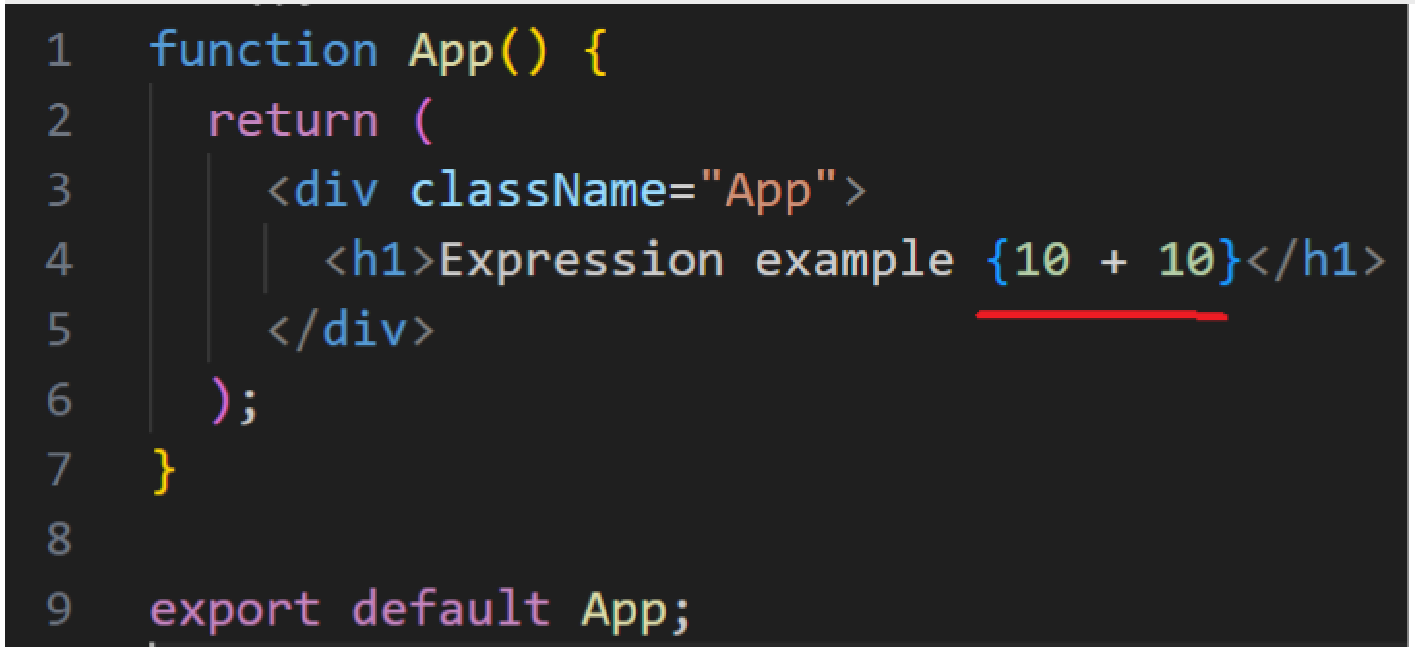
-A single DOM element, like a <div>, <h1>, or any other HTML element.

-A **fragment** which is a way to return multiple elements without wrapping them in a parent element. We can use an empty tag (<> ... </>) or the <React.Fragment>

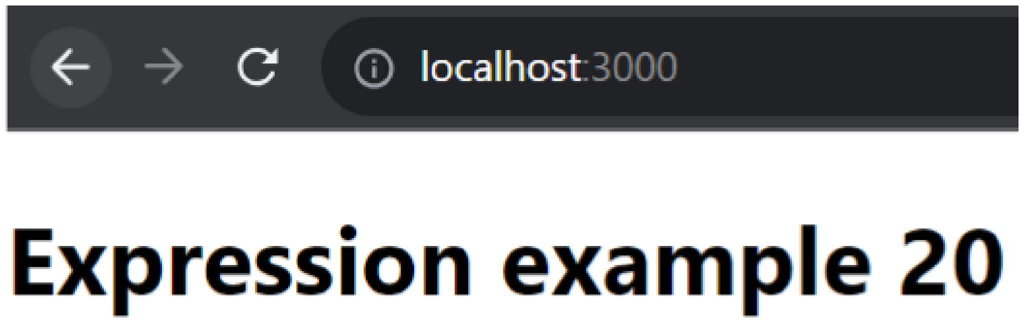
-Nested elements.

What is JSX?

Stands for **JavaScript XML**. It is an extension to JavaScript used in React. It allows us to write HTML in JavaScript. Using curly braces { }, we can write JavaScript **expressions** in JSX.



```
function App() {
  return (
    <div className="App">
      <h1>Expression example {10 + 10}</h1>
    </div>
  );
}

export default App;
```



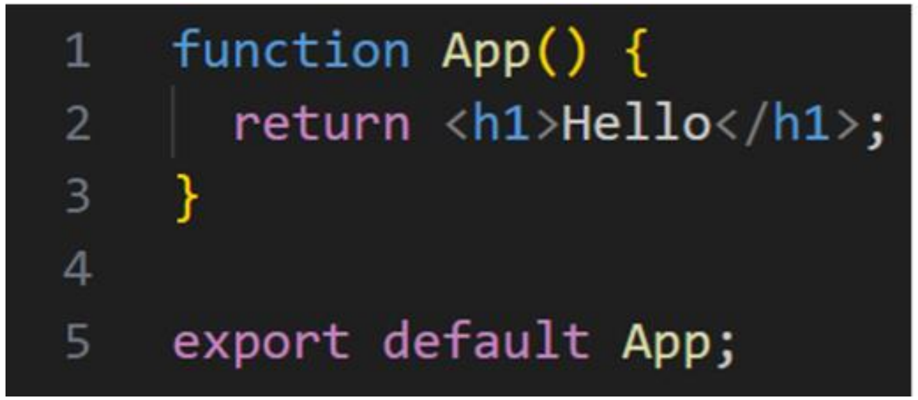
localhost:3000

Expression example 20

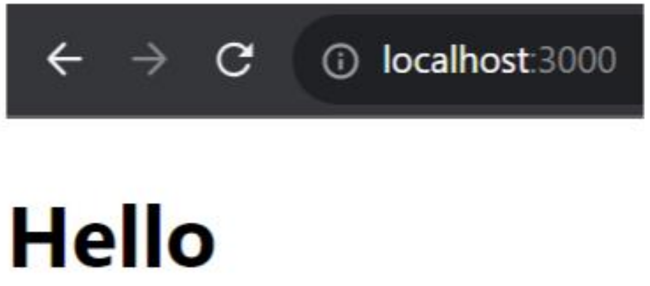
How does the browser understand and read JSX?

It does not. React uses **Babel** which is a transpiler(a compiler that translates one form of syntax into another) that transforms JSX into regular JavaScript object before passing it to the browser. Transpiling occurs during the build process and runtime. The browser receives a tree of objects that have been described using the React API.

What is this below? Does it return HTML? What does it compile to?

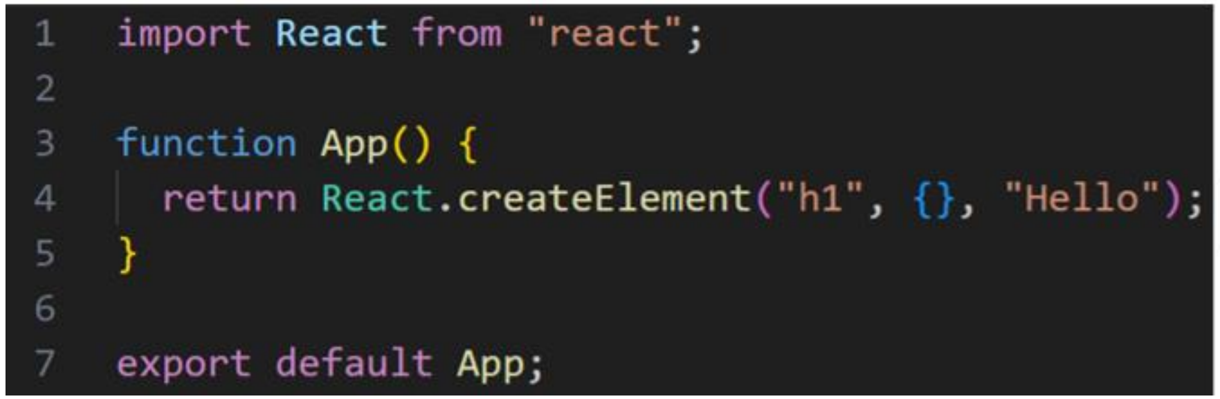


```
function App() {
  return <h1>Hello</h1>;
}

export default App;
```

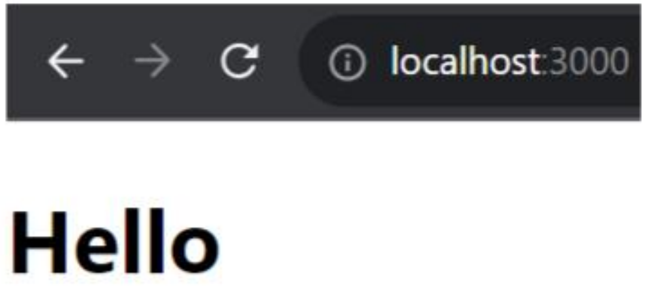


Hello

App is a functional component that uses **JSX** syntax and renders Hello in the browser. The h1 tag rendered by the **App** component is a pure JavaScript function call to **React.createElement().** So the above example would compile(using Babel) into the code below before being used by the browser:



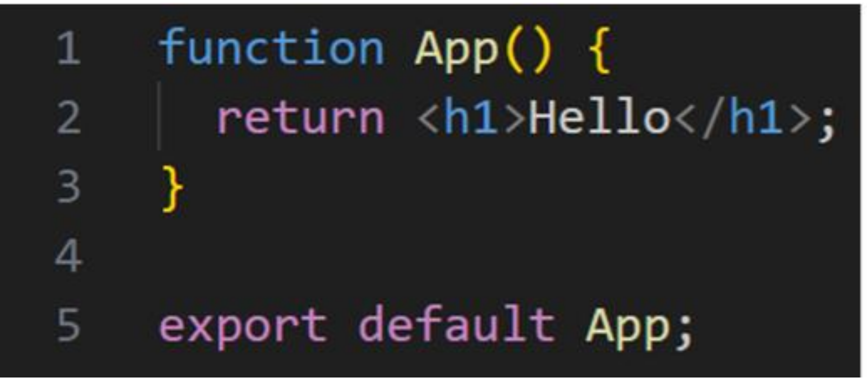
```
import React from "react";

function App() {
  return React.createElement("h1", {}, "Hello");
}

export default App;
```



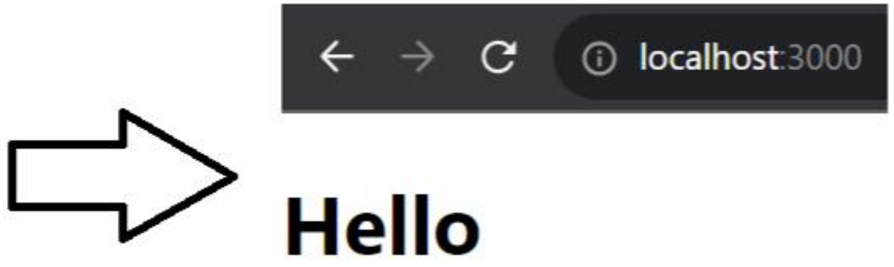
Hello

React.createElement() is used to create React elements.

So without **JSX** we have to use **React.createElement()** to create React elements. But In modern React development, we typically use **JSX** syntax like below:



```
function App() {
  return <h1>Hello</h1>;
}

export default App;
```



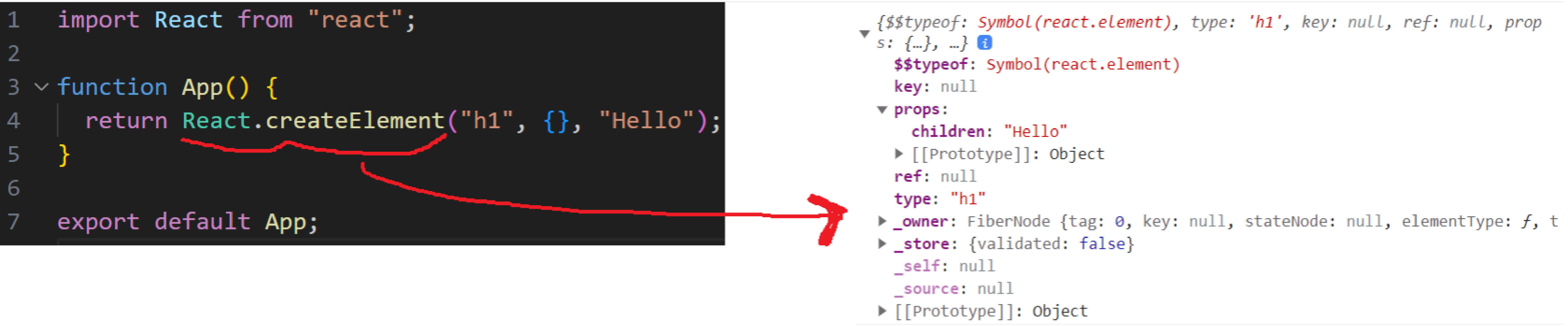
which is more readable and expressive than manually creating elements with **React.createElement()**

In the question, the App component returns JSX not HTML.

As a result **JSX** gets transpiled into calls to **React.createElement()** behind the scenes during the build process and runtime, making it easier for developers to work with React. When the JSX is transpiled (using a tool like Babel), it compiles to calls to **React.createElement()** automatically, which creates JavaScript objects representing the virtual DOM elements. These JavaScript objects representing the virtual DOM elements are used by React to update the actual DOM efficiently.

What does React.createElement() function return?

It returns an object like below. These objects are known as React elements, but they are just plain JavaScript objects. These objects describe what you want to see on the screen.



```
import React from "react";

function App() {
  return React.createElement("h1", {}, "Hello");
}

export default App;
```

```
▼ {$$typeof: Symbol(react.element), type: 'h1', key: null, ref: null, prop
  s: {…}, …}
    $$typeof: Symbol(react.element)
    key: null
  ▼ props:
      children: "Hello"
    ▶ [[Prototype]]: Object
    ref: null
    type: "h1"
  ▶ _owner: FiberNode {tag: 0, key: null, stateNode: null, elementType: ƒ, t
  ▶ _store: {validated: false}
    _self: null
    _source: null
  ▶ [[Prototype]]: Object
```

These objects represent **HTML** elements and don't live on the page(the real DOM) but on the **virtual DOM**. React reads these objects and uses them to create HTML elements on the virtual DOM and they get synced with the real DOM.

So there are trees of objects on the virtual DOM and trees of objects on the real DOM. React updates the associated DOM element when we change data on a React element.

$$typeof: Identifies the object as a React element. It is used for protection against Cross-site Scripting (XSS) attacks.

type: Type of React element to be rendered. E.g h1,div

key: Used to uniquely identify elements among siblings while mapping over an array.

props: Object containing properties. "props" in React

children: Children you want to pass into the element. When adding multiple children, we use array.

ref: Reference to an actual DOM node. Allows you to get direct access to a DOM element.

Also note that From React 17 the Facebook team collaborated with Babel.

Because JSX was compiled into **React.createElement()** calls, you had to have **React** imported in scope if you used JSX. With the new transform, you can skip the (required) import React from 'react' statement in each component file. It is possible to write JSX without importing the React library at the top level or having React in scope. However, we would still need to import React to use Hooks and other React exports.