

JSX stands for JavaScript XML.

WHAT IS JSX?

JSX

- 👉 Declarative syntax to describe what components **look like** and **how they work**

Component

- Data
- Logic
- Appearance

©danny

JSX is a declarative syntax which is used to describe what components look like and how they work based on their data and logic.

WHAT IS JSX?

JSX

- 👉 Declarative syntax to describe what components **look like** and **how they work**
- 👉 Components must **return** a block of JSX

```
function Question(props) {
  const question = props.question;
  const [upvotes, setUpvotes] = useState(0);

  const upvote = () => setUpvotes((v) => v + 1);

  const openQuestion = () => {}; // Todo

  return (
    <div>
      <h4 style={{ fontSize: "2.4rem" }}>
        {question.title}
      </h4>
      <p>{question.text}</p>
      <p>{question.hours} hours ago</p>
      <UpvoteBtn onClick={upvote} />
      <Answers
        numAnswers={question.num}
        onClick={openQuestion}
      />
    </div>
  );
}
```

JSX returned from component

©danny

Each component should return a block of JSX which is used by REACT to render the component on the UI.

WHAT IS JSX?

JSX

- 👉 Declarative syntax to describe what components look like and how they work
- 👉 Components must return a block of JSX
- 👉 Extension of JavaScript that allows us to embed JavaScript, CSS and React components into HTML

```
function Question(props) {  
  const question = props.question;  
  const [upvotes, setUpvotes] = useState(0);  
  
  const upvote = () => setUpvotes((v) => v + 1);  
  
  const openQuestion = () => {}; // Todo  
  
  return 

<h4 style={{ fontSize: "2.4rem" }}>  
      {question.title}</h4>  
    <p>{question.text}</p>  
    <p>{question.hours}</p>  
    <UpvoteBtn onClick={upvote} />  
    <Answers  
      numAnswers={question.num}</Answers>  
    <openQuestion />  
  </div>  
};


```

JSX returned from component

JSX is an extension of JAVASCRIPT which allows us to combine parts of HTML, CSS and JAVASCRIPT all into one block of code. We can write HTML and embed some piece of JAVASCRIPT when necessary.

WHAT IS JSX?

JSX

- 👉 Declarative syntax to describe what components look like and how they work
- 👉 Components must return a block of JSX
- 👉 Extension of JavaScript that allows us to embed JavaScript, CSS, and React components into HTML
- 👉 Each JSX element is converted to a `React.createElement` function call
- 👉 We could use React without JSX

```
<header>  
  <h1 style="color: red">  
    Hello React!  
  </h1>  
</header>
```



BABEL

```
React.createElement(  
  'header',  
  null,  
  React.createElement(  
    'h1',  
    { style: { color: 'red' } },  
    'Hello React!'</pre>
```



Hello React!



Q: If REACT is a JAVASCRIPT library, how does it understand this JSX?
A: There is a simple way of converting JSX into JAVASCRIPT. This is done by a tool called a BABEL which gets automatically included into our application by CREATE-REACT-APP.

Each JSX code gets converted into `React.createElement()` function call. This conversion is necessary because browsers do not understand JSX. They only understand HTML. All the JSX gets converted into many nested `React.createElement()` function calls behind the scenes. These function calls create the HTML elements that we see on the screen.

We can use REACT without JSX. We can manually create `React.createElement()` function calls instead of JSX.

1. This makes the code very hard to read and understand.
2. Everyone uses the JSX way.

JSX IS DECLARATIVE

IMPERATIVE

- Manual DOM element selections and DOM traversing
- Step-by-step DOM mutations until we reach the desired UI

JS

```
const title = document.querySelector("title");
const upvoteBtn = document.querySelector("btn");
title.textContent = [0] ? question.title;
let upvotes = 0;
upvoteBtn.addEventListener("click", function(){
  upvotes++;
  title.textContent =
    [upvotes] ? question.title;
  title.classList.add("upvoted");
});
```

When we build UI using vanilla JAVASCRIPT, we are using imperative approach by default. Imperative approach basically tells how to do things.

1. We manually select DOM, traverse the DOM and attach event handlers to elements.
2. If something happens in the app, like click on button we give the browser step by step instructions on how to mutate those DOM elements until we reach the desired and updated UI.

In a complex application like AIRBNB it is not feasible.

JSX IS DECLARATIVE

IMPERATIVE

"How to do things"

JS

- Manual DOM element selections and DOM traversing
- Step-by-step DOM mutations until we reach the desired UI

```
const title = document.querySelector("title");
const upvoteBtn = document.querySelector("btn");
title.textContent = [0] ${question.title};
let upvotes = 0;
upvoteBtn.addEventListener("click", function(){
  upvotes++;
  title.textContent =
    [${upvotes}] ${question.title};
  title.classList.add("upvoted");
});
```

DECLARATIVE

"What we want"



- Describe what UI should look like using JSX, **based on current data**
- React is an **abstraction** away from DOM: **we never touch the DOM**
- Instead, we think of the UI as a **reflection of the current data**

```
function Question(props) {
  const question = props.question;
  const [upvotes, setUpvotes] = useState(0);
  const upvote = () => setUpvotes(v => v + 1);

  return (
    <div>
      <h4>question.title</h4>
      <p>question.text</p>
      <UpvoteBtn
        onClick={upvote}
        upvotes={upvotes}
      />
    </div>
  );
}
```

1. A declarative approach describes what UI should look like. We use the JSX to describe the UI based on current data.
2. All this happens without any DOM manipulation. REACT is an abstraction away from DOM.
3. We think of UI as reflection of current data and let REACT automatically synchronize the UI with that data.