# Introducing JSX:

# It is a syntax extension to JavaScript. It has full power of JavaScript.

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

React embraces the fact that rendering logic is inherently coupled with other UI logic: how events are handled, how the state changes over time, and how the data is prepared for display.

React [doesn’t require](https://reactjs.org/docs/react-without-jsx.html) using JSX, but its helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages.

### Embedding Expressions in JSX

In the example below, we declare a variable called name and then use it inside JSX by wrapping it in curly braces:

const name = 'Josh Perez';

const element = <h1>Hello, {name}</h1>;

ReactDOM.render(

element,

document.getElementById('root')

);

You can put any valid [JavaScript expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#Expressions) inside the curly braces in JSX. For example, 2 + 2, user.firstName, or formatName(user) are all valid JavaScript expressions.

React DOM uses camelCase property naming convention instead of HTML attribute names.

For example, **class** becomes **[className](https://developer.mozilla.org/en-US/docs/Web/API/Element/className" \t "_blank)** in JSX, and **tabindex** becomes **[tabIndex](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/tabIndex" \t "_blank)**.

### JSX Prevents Injection Attacks

It is safe to embed user input in JSX:

const title = response.potentiallyMaliciousInput;

// This is safe:

const element = <h1>{title}</h1>;

By default, React DOM [escapes](https://stackoverflow.com/questions/7381974/which-characters-need-to-be-escaped-on-html) any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that’s not explicitly written in your application. Everything is converted to a string before being rendered. This helps prevent [XSS (cross-site-scripting)](https://en.wikipedia.org/wiki/Cross-site_scripting) attacks.

### JSX Represents Objects

Babel compiles JSX down to React.createElement() calls.

These two examples are identical:

const element = (

<h1 className="greeting">

Hello, world!

</h1>

);

const element = React.createElement(

'h1',

{className: 'greeting'},

'Hello, world!'

);

React.createElement() performs a few checks to help you write bug-free code but essentially it creates an object like this:

// Note: this structure is simplified

const element = {

type: 'h1',

props: {

className: 'greeting',

children: 'Hello, world!'

}

};

# [Rendering an Element/Component into the DOM:](https://reactjs.org/docs/rendering-elements.html)

In index.html,

<div id=’root’></div>

In App.js,

*1. Rendering an element:*

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

ReactDOM.render(element, document.getElementById('root'));

*2. Rendering Component*

ReactDOM.render(<MyComponent />, document.getElementById('root'));

## 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](https://reactjs.org/redirect-to-codepen/rendering-elements/update-rendered-element) with the browser tools:

function tick() {

const element = (

<div>

<h1>Hello, world!</h1>

<h2>It is {new Date().toLocaleTimeString()}.</h2>

</div>

);

ReactDOM.render(element, document.getElementById('root'));

}

setInterval(tick, 1000);

# Components

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

There are two types of components in React.

**Function Component:**

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**Class Component:**

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

**Props:**

React allows us to pass information to a Component using something called **props**(stands for properties). These are Read-Only. We cannot change the value.

For example, this code renders “Hello World!” on the page:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

const element = <Welcome name="Hello World!" />;

ReactDOM.render(

element,

document.getElementById('root')

);

**State:**

State allows React components to change their output over time in response to user actions, network responses, and anything else, without violating this rule.