[Introducing JSX](https://reactjs.org/docs/introducing-jsx.html)

# JSX produces React “elements”

# React [separates concerns](https://en.wikipedia.org/wiki/Separation_of_concerns) with loosely coupled units called “components”

# React [doesn’t require](https://reactjs.org/docs/react-without-jsx.html) using JSX,

# 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.

**e.g const name = 'Josh Perez';**

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

# After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions

# You may also use curly braces to embed a JavaScript expression in an attribute

# 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.

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

**const element = (**

**<h1 className="greeting">**

**Hello, world!**

**</h1>**

**);**

**OR**

**const element = React.createElement(**

**'h1',**

**{className: 'greeting'},**

**'Hello, world!'**

**);**

React.createElement() essentially it creates an object like this:

**// Note: this structure is simplified**

**const element = {**

**type: 'h1',**

**props: {**

**className: 'greeting',**

**children: 'Hello, world!'**

**}**

**};**

These objects are called “React elements”. React reads these objects and uses them to construct the DOM and keep it up to date.

[Rendering Elements](https://reactjs.org/docs/rendering-elements.html)

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

# React elements are [immutable](https://en.wikipedia.org/wiki/Immutable_object). Once you create an element, you can’t change its children or attributes.

# With our knowledge so far, the only way to update the UI is to create a new element, and pass it to [ReactDOM.render()](https://reactjs.org/docs/react-dom.html#render).

**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);**

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.

# 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.

#

#

#

#

#

#

#

#

#

#

#