# Why React?

In the current lecture, the speaker, Katie, a Software Engineer at Meta, discusses the significance of React in the tech industry. Key points include:

React's Popularity: It's widely used for building user interfaces due to its ease of learning and flexibility.

Composition vs. Inheritance: Unlike traditional object-oriented programming, React uses composition, which can be initially challenging but is supported by extensive documentation and community resources.

Custom UIs: React allows for the creation of highly customizable user interfaces, making it suitable for complex applications.

Code Re-usability: React promotes reusing components, which enhances efficiency and consistency across applications.

Comparison with Angular: React is a client-side library that requires integration with other libraries, while Angular is a full-fledged framework that provides more out-of-the-box solutions.

Learning Approach: Beginners are encouraged to start with simple projects and gradually build complexity, utilizing best practices and documentation.

This lecture emphasizes the importance of React skills for a successful career in tech. Would you like to explore any specific aspect of React further?

# React.js overview

Component-based architecture is like building a house using different blocks. Each block, or component, has its own specific purpose and can be reused in different parts of the house without needing to change the other blocks. For example, think of a website as a collection of these blocks: you might have a header block with a logo and navigation, a footer block with contact information, and a content block that displays articles. Each of these blocks can work independently, making it easier for multiple developers to collaborate on the same project without stepping on each other's toes.

This approach not only makes the development process smoother but also helps keep the code organized and manageable. Just like how you can rearrange blocks to create different layouts for your house, in React, you can combine components in various ways to create a unique user interface. This flexibility is one of the reasons why React is so popular among developers!

# Introduction to functional components

Functional components in React, which are similar to traditional JavaScript functions. Here are the key points:

Functional Components: These are reusable blocks of code that take inputs, perform actions, and return outputs, much like JavaScript functions.

JSX: React uses a syntax called JavaScript XML (JSX), which resembles HTML but allows you to write JavaScript code within it. JSX is used to create dynamic content in React components.

Creating Components: To create a functional component, you define a JavaScript function with a capitalized name. Inside the function, you can return JSX, which can include variables wrapped in curly brackets to display dynamic content.

Rendering: A React component must be used as a JSX element to render it on the webpage. The process of converting JSX to HTML is called transpiling.

This lecture sets the foundation for understanding how to build user interfaces in React using functional components and JSX.

# Creating React components

* Components are reusable blocks of code in React.
* You started by using Create React App to set up a new application.
* The app.js file was modified to create a simple functional component named header that displays "Hello World" in an H1 element.
* You learned how to render this component by calling it within the app function using JSX syntax.
* The importance of isolating components into their own files for reusability was also highlighted.
* This foundational knowledge is essential for building scalable and maintainable applications in React.

function Heading() {

return <h1>hello world</h1>;

}

function App() {

return <Heading />;

}

export default App;

# Transpiling JSX

## Introduction

Components are a nice way to build websites in React because they allow you to build more modular apps. However, how do you build components using React, JSX, and JavaScript? You'll learn how this works in this lesson item.

A browser cannot understand JSX syntax.

This means that making a browser understand React code requires a lot of supporting technologies.

An example of such a technology is a transpiler.

A transpiler takes a piece of code and transforms it into some other code.

To understand why this is done, here is an example of an ES6 variable declaration:

1

const PI = 3.14

This is perfectly valid ES6 syntax.

However, if you were using a very old computer, that computer will have an old browser. Perhaps that browser was built before ES6 came out in 2015.

This means that the JavaScript engine that is built into your old computer's browser is likely to be an ES5 JavaScript engine.

In ES5, the only way to declare a variable is the following:

1

var pi = 3.14

What this means is that for this old browser to understand the ES6 code, the only way to do it is by transpiling it.

This means you need a way of essentially translating the modern js code into a format that an older js engine can understand.

Let's say that you want to use a brand new, most modern ECMAScript syntax in an app. The only problem is that this new syntax is currently not supported by any browser; even an up-to-date browser.

However, by transpiling the new most-modern JavaScript syntax into something that modern browsers can understand, it is able to convert some code that the browser cannot comprehend, into code that it can comprehend, run, and produce a result from.

Likely the most popular site that shows off how this works is [Babel](https://babeljs.io/" \t "https://www.coursera.org/learn/react-basics/supplement/vjUf1/_blank). As the heading of the website reads, "Babel is a JavaScript Compiler".

This finally brings you to the point of this discussion about transpiling JavaScript code.

What Babel does is this: it allows you to transpile JSX code (which cannot be understood by a browser) into plain JavaScript code (which can be understood by a browser).

This is where React and JSX come in.

For React code to be understood by a browser, you need to have a transpiling step in which the JSX code gets converted to plain JavaScript code that a modern browser can work with.

To demonstrate how this works, let’s use the Heading component from the previous lesson.

Add the JSX code into [the online Babel repl](https://babeljs.io/repl" \l "?browsers=defaults, not ie 11, not ie_mob 11&build=&builtIns=false&corejs=3.21&spec=false&loose=false&code_lz=GYVwdgxgLglg9mABACQKYEMAmMwHMAUADgE5yEDOAlIgN4BQijixqUIxSAPABYCMAfDRJlyAOlhQANqgC-nAPR9-dGUA&debug=false&forceAllTransforms=false&modules=false&shippedProposals=false&circleciRepo=&evaluate=false&fileSize=false&timeTravel=false&sourceType=module&lineWrap=true&presets=env,react,stage-2&prettier=false&targets=&version=7.22.20&externalPlugins=&assumptions={}" \o "Link updated to use the latest version of Babel" \t "https://www.coursera.org/learn/react-basics/supplement/vjUf1/_blank). Repl stands for "read-eval-print loop" and it accepts code you write, evaluates it, and produces some result. In the specific case of [the online Babel repl](https://babeljs.io/repl" \l "?browsers=defaults, not ie 11, not ie_mob 11&build=&builtIns=false&corejs=3.21&spec=false&loose=false&code_lz=GYVwdgxgLglg9mABACQKYEMAmMwHMAUADgE5yEDOAlIgN4BQijixqUIxSAPABYCMAfDRJlyAOlhQANqgC-nAPR9-dGUA&debug=false&forceAllTransforms=false&modules=false&shippedProposals=false&circleciRepo=&evaluate=false&fileSize=false&timeTravel=false&sourceType=module&lineWrap=true&presets=env,react,stage-2&prettier=false&targets=&version=7.22.20&externalPlugins=&assumptions={}" \o "Link updated to use the latest version of Babel" \t "https://www.coursera.org/learn/react-basics/supplement/vjUf1/_blank), that result is some transpiled code. Here's a more detailed explanation.

If you've visited the above-linked URL, you'll find a web page that has two panels. On the left, there's source JSX code:

1

2

3

function Heading(props) {

    return <h1>{props.title}</h1>

}

... and on the right, there's the transpiled, plain JavaScript code. However, ensure that you select the classic runtime for React in the left sidebar.

1

2

3

function Heading(props) {

  return /\*#\_\_PURE\_\_\*/React.createElement("h1", null, props.title);

}

If you now analyze the difference between the source JSX code and the transpiled, plain JavaScript code, dis-regarding the comment, here's the body of the Heading function:

1

2

React.createElement("h1", null, props.title);

So, here you have a React object, and this object has a createElement() method on it. The method is invoked with three arguments:

"h1"

null

props.title

The first argument is the DOM element to render - in this case, an h1 element. The second property is any HTML attribute that should be added, and there's a null here - meaning, there should be an object with some data, but there isn't any data so instead of the object there's the null value. The third property is the contents of the inner HTML of the DOM element specified as the first argument - in this case, the contents of the inner HTML of the h1 element.

Now let’s use Babel again, and this time transpile the render syntax for the Heading component:

1

<Heading title="This is the heading text!"></Heading>

Again using [the Babel repl](https://babeljs.io/repl" \l "?browsers=defaults, not ie 11, not ie_mob 11&build=&builtIns=false&corejs=3.21&spec=false&loose=false&code_lz=DwCQpghgJglgdgcwAQBcYoDZgLwCIAqAFjAM5KmqFhJXTzIpgAeKAhLgHzAD04diHIA&debug=false&forceAllTransforms=false&modules=false&shippedProposals=false&circleciRepo=&evaluate=false&fileSize=false&timeTravel=false&sourceType=module&lineWrap=true&presets=env,react,stage-2&prettier=false&targets=&version=7.22.20&externalPlugins=&assumptions={}" \o "Link updated to use the latest Babel" \t "https://www.coursera.org/learn/react-basics/supplement/vjUf1/_blank), and as can be confirmed in [the link](https://babeljs.io/repl" \l "?browsers=defaults, not ie 11, not ie_mob 11&build=&builtIns=false&corejs=3.21&spec=false&loose=false&code_lz=DwCQpghgJglgdgcwAQBcYoDZgLwCIAqAFjAM5KmqFhJXTzIpgAeKAhLgHzAD04diHIA&debug=false&forceAllTransforms=false&shippedProposals=false&circleciRepo=&evaluate=false&fileSize=false&timeTravel=false&sourceType=module&lineWrap=true&presets=env,react,stage-2&prettier=false&targets=&version=7.18.5&externalPlugins=&assumptions={}" \t "https://www.coursera.org/learn/react-basics/supplement/vjUf1/_blank), the output of the tranpilation is the following code. Ensure that you select the classic runtime for React in the left sidebar.

1

2

3

4

/\*#\_\_PURE\_\_\*/

React.createElement(Heading, {

  title: "This is the heading text!"

});

Again, you have the React.createElement() method call, and this time, the first item to render is Heading, and then you have an object as the second argument (instead of a null that you had in the previous transpilation example).

This brings me to an interesting question: What is the minimum code that a component must have to be able to show something on the screen when rendered?

You can see the answer below:

1

2

3

4

function Example() {

    return <div>An element</div>

}

export default Example