1. [React](https://reactjs.org/) is a library for building user interfaces. React is not a framework – it's not even exclusive to the web.
2. It's used with other libraries to render to certain environments. For instance, [React Native](https://reactnative.dev/) can be used to build mobile applications; [React 360](https://facebook.github.io/react-360/) can be used to build virtual reality applications;
3. Main features of react are Virtual DOM, JSX, components
   1. **Virtual DOM**

The entire structure of a web page can be represented using the DOM (Document Object Model) - an organization of html elements that we can manipulate, modify, delete, or add new ones. JavaScript is used to interact with the DOM. However, when we try to manipulate html elements using JavaScript, we may experience performance degradation, especially when changing a large number of elements. And operations on elements can take some time, which will inevitably affect the user experience. However, if we were working from js code with JavaScript objects, the operations would be performed faster.

To solve the performance problem, the concept of a virtual DOM just appeared.

**3.1.1** A virtual DOM is a lightweight copy of a regular DOM. And a distinctive feature of React is that this library works with a virtual DOM, not a regular one.

If the application needs to find out information about the state of the elements, then the virtual DOM is accessed.

If you need to change the elements of a web page, the changes are first made to the virtual DOM. Then the new state of the virtual DOM is compared to the current state. And if these states differ, React finds the minimum number of manipulations that are needed before updating the real DOM to a new state and performs them.

As a result, this scheme of interaction with web page elements works much faster and more efficiently than if we were working directly from JavaScript with the DOM.

Other React Features

* 1. **JSX**.
     1. JSX is a combination of JavaScript and XML code and provides a simple and intuitive way to define visual interface code.

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

**3.2.2** 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.

1. **Rendering Elements**

Elements are the smallest building blocks of React apps.

An element describes what you want to see on the screen:

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

Unlike browser DOM elements, React elements are plain objects, and are cheap to create. React DOM takes care of updating the DOM to match the React elements.

React elements are [immutable](https://en.wikipedia.org/wiki/Immutable_object). Once you create an element, you can’t change its children or attributes. An element is like a single frame in a movie: it represents the UI at a certain point in time.

# Components and Props

Another distinctive feature of the library is the concentration on components - we can create individual components and then easily transfer them from project to project.

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

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

**5.1** The simplest way to define a component is to write a JavaScript function:

function Welcome(props) {

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

}

This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element. Such components called “function components” because they are literally JavaScript functions.

**5.2** Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.

## Props are Read-Only

Whether you declare a component [as a function or a class](https://reactjs.org/docs/components-and-props.html#function-and-class-components), it must never modify its own props.

Such functions are called [“pure”](https://en.wikipedia.org/wiki/Pure_function) because they do not attempt to change their inputs, and always return the same result for the same inputs.

In contrast, there are **impure** functions because they change their own input:

React is pretty flexible but it has a single strict rule:

**All React components must act like pure functions with respect to their props.**

Of course, application UIs are dynamic and change over time.

# Handling Events

Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences:

* React events are named using ***camelCase***, rather than lowercase.
* With JSX you pass a ***function*** as the event handler, rather than a string.
* Another difference is that you cannot return false to prevent default behavior in React. You must call ***preventDefault*** explicitly.

When using React, you generally don’t need to call addEventListener to add listeners to a DOM element after it is created. Instead, just provide a listener when the element is initially rendered.

# Conditional Rendering

In React, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state of your application.

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like [if](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else) or the [conditional operator](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Conditional_Operator) to create elements representing the current state, and let React update the UI to match them.

**7.1**

We’ll create a Greeting component that displays either of these components depending on whether a user is logged in:

function Greeting(props) {

const isLoggedIn = props.isLoggedIn;

if (isLoggedIn) { return <UserGreeting />; } return <GuestGreeting />;}

ReactDOM.render(

// Try changing to isLoggedIn={true}:

<Greeting isLoggedIn={false} />, document.getElementById('root'));

This example renders a different greeting depending on the value of **isLoggedIn** prop.

### **7.2 Inline If-Else with Conditional Operator**

Another method for conditionally rendering elements inline is to use the JavaScript conditional operator [condition ? true : false](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Conditional_Operator).

In the example below, we use it to conditionally render a small block of text.

render() {

const isLoggedIn = this.state.isLoggedIn;

return (

<div>

The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in. </div>

);

}

## **Keys**

**8.0.1** Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity:

**8.0.2.**

const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) =>

<li key={number.toString()}> {number}

</li>

);

The best way to pick a key is to use a string that uniquely identifies a list item among its siblings. For example< it can be ID or NAME or something similar