React.js is a powerful, component-based JavaScript library for building user interfaces. It is maintained by Meta and a massive community of developers. This crash course will take you from the fundamental concepts to building a functional application.

---

1. What is React?

React is often called a framework, but it is technically a **library**. It focuses on the "View" layer of an application. The two biggest selling points are:

- **Components:** Small, reusable pieces of UI (like a button or a navigation bar).
- **Declarative Syntax:** You describe *what* the UI should look like based on the current state, and React handles updating the DOM to match.

---

2. Essential Prerequisites

Before diving in, ensure you are comfortable with:

- **HTML/CSS:** Semantic tags and Flexbox/Grid.
- **Modern JavaScript (ES6+):** Arrow functions, Destructuring, Spread operator, and Template literals.
- **NPM/Node.js:** Installed on your system to manage packages.

---

3. Setting Up Your First Project

The modern way to start a React project is using **Vite**. It is significantly faster than the older create-react-app.

1. Open your terminal and run:
   Bash
   `npm create vite@latest my-react-app -- --template react`

2. Navigate into the folder: cd my-react-app
3. Install dependencies: npm install
4. Start the dev server: npm run dev

---

4. JSX: JavaScript XML

JSX allows you to write HTML-like code directly inside JavaScript. It makes the code more readable and visual.

Key Rules of JSX:

- You must return a **single parent element** (or use a Fragment: <> ... </>).
- Use className instead of class.
- Use curly braces {} to embed JavaScript variables or expressions.

JavaScript

```
function Welcome() {
  const name = "Developer";
  return (
    <div className="container">
      <h1>Hello, {name}!</h1>
      <p>Today is {new Date().toLocaleDateString()}</p>
    </div>
  );
}
```

---

5. Components & Props

Components are functions that return JSX. **Props** (short for properties) are how you pass data from a parent component to a child.

Parent Component:

JavaScript

```
import Profile from './Profile';

function App() {
  return (
    <Profile name="Alice" job="Engineer" />
  );
}
```

Child Component (Profile.jsx):

JavaScript

```
function Profile(props) {
  return (
    <div>
      <h2>{props.name}</h2>
      <p>Role: {props.job}</p>
    </div>
  );
}
```

---

6. State & Hooks (useState)

State is data that changes over time. When state updates, React **re-renders** the component to show the new data. We use the useState Hook to manage this.

JavaScript

```
import { useState } from 'react';

function Counter() {
  // count = current value, setCount = function to update it
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Increase
      </button>
    </div>
  );
}
```

---

7. Handling Side Effects (useEffect)

The useEffect Hook allows you to perform "side effects" like fetching data, manually changing the DOM, or setting up subscriptions.

JavaScript

```
import { useState, useEffect } from 'react';

function DataFetcher() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch('https://api.example.com/items')
      .then(res => res.json())
      .then(json => setData(json));
  }, []); // Empty array means this runs ONLY once on mount
}
```

---

8. Lists and Keys

To render multiple items, we use the JavaScript .map() method. You must always provide a unique key prop to help React track which items changed.

JavaScript

```
const users = [
  { id: 1, name: 'John' },
  { id: 2, name: 'Jane' }
];
```

```
function UserList() {
  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

---

9. Summary Table: Core Concepts

Feature	Description
JSX	Syntax extension that allows HTML in JS.
Components	Independent, reusable UI building blocks.
Props	Read-only data passed from parent to child.
State	Internal data that triggers a re-render when changed.
Hooks	Functions (like useState) that "hook" into React features.
Virtual DOM	A lightweight copy of the real DOM used for performance.

Next Steps

To truly master React, you should build a small project.