Here's a simple and beginner-friendly guide to **React.js**, covering key concepts and components you'll need to start your React journey!

What is React?

React is a **JavaScript library** used for building user interfaces (UIs), particularly for single-page applications where you need fast, interactive user experiences. It allows you to create reusable UI components and efficiently update the UI when data changes.

Key Concepts of React:

1. **JSX (JavaScript XML)**:
   - JSX is a syntax extension for JavaScript that allows you to write HTML-like code inside JavaScript. React components are often written using JSX.
   - **Example**:

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

2. **Components**:
   - React components are reusable pieces of code that return JSX and define how parts of the UI should look and behave.
   - **Types of Components**:
     - **Functional Components**: Functions that return JSX.

       ```
       Jsx code:-
       function Welcome(props) {
         return <h1>Hello, {props.name}</h1>;
       }
       ```

     - **Class Components**: ES6 classes that extend `React.Component` and can have additional features like state and lifecycle methods.

       ```
       Jsx code:-
       class Welcome extends React.Component {
         render() {
           return <h1>Hello, {this.props.name}</h1>;
         }
       }
       ```

3. **Props (Properties)**:
   - **Props** are inputs to a component and are passed from parent components to child components.
   - Props are **read-only** and cannot be modified by the child component.
   - **Example**:

     ```
     Jsx code
     function Welcome(props) {
       return <h1>Hello, {props.name}</h1>;
     ```

```
}

// Usage
<Welcome name="John" />
```

4. **State**:
   - **State** is used to manage data that changes over time, typically in response to user interactions. Unlike props, state is **mutable** and can be changed within a component.
   - In **functional components**, you use the `useState` hook.
   - **Example** (Functional Component with State):

```
Jsx code
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

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

5. **Event Handling**:
   - React uses event handlers to capture user input or actions.
   - Example of a click event:

```
Jsx code

function MyButton() {
  const handleClick = () => {
    alert('Button clicked!');
  };

  return <button onClick={handleClick}>Click Me</button>;
}
```

6. **Conditional Rendering**:
   - You can render components or elements based on conditions using `if` statements, ternary operators, or logical `&&` operators.
   - Example:

```
jsx code
function Greeting({ isLoggedIn }) {
  return isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please log
in.</h1>;
}
```

7. **Lists and Keys**:
   - React allows you to render lists of elements by mapping over an array. When rendering lists, each element should have a unique `key` to help React optimize rendering.
   - Example:

     Jsx code:-

```
function ListItem(props) {
  return <li>{props.value}</li>;
}

function NumberList() {
  const numbers = [1, 2, 3, 4];
  return (
    <ul>
      {numbers.map((number, index) => (
        <ListItem key={index} value={number} />
      ))}
    </ul>
  );
}
```

8. **Forms**:
   - Handling forms in React requires managing state for input fields, so the form data is controlled by React.
   - Example:

     Jsx code:-

```
function NameForm() {
  const [name, setName] = useState('');

  const handleChange = (event) => {
    setName(event.target.value);
  };

  const handleSubmit = (event) => {
    alert('A name was submitted: ' + name);
    event.preventDefault();
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={handleChange}
/>
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}
```

9. **useEffect Hook**:

- The `useEffect` hook lets you perform side effects (like fetching data, updating the DOM, or subscribing to external services) in function components.
- It replaces lifecycle methods in class components like `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.
- Example:

```
jsx code:-
import React, { useState, useEffect } from 'react';

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds((prev) => prev + 1);
    }, 1000);

    return () => clearInterval(interval); // Cleanup
  }, []);

  return <div>Seconds: {seconds}</div>;
}
```

10. **React Router**:
    - React Router allows you to handle navigation between different pages in your app. It helps you create single-page applications with multiple views.
    - Example of using React Router:

```
Cmd terminal :-
npm install react-router-dom
jsx code:-
import { BrowserRouter as Router, Route, Link } from 'react-
router-dom';

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link> | <Link to="/about">About</Link>
      </nav>

      <Route path="/" exact>
        <h1>Home Page</h1>
      </Route>
      <Route path="/about">
        <h1>About Page</h1>
      </Route>
    </Router>
  );
}
```