1. **Set Up React Environment and create an application printing "Hello World"**
    - Install Node.js and npm.
    - Create a new React application using `create-react-app`.
    - Edit App.js to render "Hello World"
    - Start the development server and view your app in a browser.

```
> npm create  create-react-app
> create-react-app hello-world
> cd hello-world
```

or

```
npx create-react-app hello-world
cd hello-world
```

App.js (override the default content of the file with the following)

```
import React from 'react';

function App() {
  return (
    <div>
      <h1>Hello, World!</h1>
    </div>
  );
}

export default App;
```

2. **Create a React application that uses a simple functional component.**

```
npx create-react-app functional-component-example
cd functional-component-example
```

Greetings.js

```
// Greeting.js
import React from 'react';

function Greeting() {
  return (
    <div>
      <h1>Hello, World!</h1>
      <p>This is a simple functional component.</p>
    </div>
  );
}

export default Greeting;
```

App.js

```
// App.js
import React from 'react';
import './App.css';
import Greeting from './Greeting';

function App() {
  return (
    <div className="App">
      <h1>My React App</h1>
      <Greeting />
    </div>
  );
}

export default App;
```

Note :- without parent element like , <p> within return() you can can not combine html elements/components

3. **Create a React application that uses both regular function components and arrow function components**

> npx create-react-app fun-arrow-components-example
> cd fun-arrow-components-example

MyComponent1.js

```
import React from 'react';

function MyComponent1() {

  return <div>Regular Function Component</div>;
}

export default MyComponent1;
```

Or

```
import React from 'react';

const MyComponent1 = function() {

  return <div>Regular Function Component</div>;
}

export default MyComponent1;
```

MyComponent2.js

```
import React from 'react';

const MyComponent2 = ( ) => {

  return <div>Arrow Function Component</div>;
}

export default MyComponent2;
```

App.js

```
// App.js
import React from 'react';
import './App.css';
import Greeting from './Greeting';

function App() {
  return (
    <div className="App">
      <MyComponent1 />
      <MyComponent2 />
    </div>
  );
}
export default App;
```

4. **<u>Create a React application that uses a simple class-based component.</u>**

```
npx create-react-app class-component-example
cd class-component-example
```

// GreetingClass.js

```
// GreetingClass.js
import React, { Component } from 'react';

class GreetingClass extends Component {
  render() {
    return (
      <div>
        <h1>Hello, World!</h1>
        <p>This is a simple class-based component.</p>
      </div>
    );
  }
}

export default GreetingClass;
```

```
// App.js
import React from 'react';
import './App.css';
```

```
import GreetingClass from './GreetingClass';

function App() {
  return (
    <div className="App">
      <h1>My React App</h1>
      <GreetingClass />
    </div>
  );
}

export default App;
```

Note:-

```
import React,{Component} from 'react';
```

Vs

```
import React,{Component} from 'react';
```

`React`: This is the default export from the 'react' module. It typically refers to the main React object and is commonly used for creating and working with React elements and components.

`{ Component }`: This part of the import statement uses destructuring to import a specific named export from the 'react' module

```
import React,{Component} from 'react';
class  HelloWorldClass extends Component { }
```

Or

```
import React from 'react';
class  HelloWorldClass extends React.Component {}
```

5. **Create a React application that has Compose multiple components within a parent component.**

```
npx create-react-app multiple-component-example
cd multiple-component-example
```

ChildComponent1.js

```
import React from 'react';

const ChildComponent1 = () => {
  return (
    <div>
      <h3>Child Component 1</h3>
    </div>
  );
};

export default ChildComponent1;
```

ChildComponent2.js

```
import React from 'react';

const ChildComponent2 = ( ) => {
  return (
    <div>
      <h3>Child Component 2</h3>
    </div>
  );
};

export default ChildComponent2;
```

ParentComponent.js

```
import React from 'react';
import ChildComponent1 from './ChildComponent1';
import ChildComponent2 from './ChildComponent2';

const ParentComponent = () => {

  return (
    <div>
      <h2>Parent Component</h2>
      <ChildComponent1 />
      <ChildComponent2 />
    </div>
  );
};

export default ParentComponent;
```

App.js

```
import React from 'react';
import ParentComponent from './ParentComponent';
import './App.css';

function App() {
  return (
    <div className="App">
      <ParentComponent />
    </div>
  );
}

export default App;
```

6. **Create a React application that uses Props in Functional Components**

Note:-

In React, props (short for "properties") are used to pass data from parent components to child components. They allow you to make your components reusable and dynamic by providing a way to customize the content and behavior of child components based on values passed from their parent components.

Parent Component: The parent component is responsible for rendering and defining the props that it will pass down to its child components.

Child Component: The child component receives the props passed down from its parent and can use these props to render content or execute functions.

Passing Props: You pass props by including them as attributes when you render a child component within a parent component.

```
npx create-react-app prop-example
cd prop-example
```

ChildComponent.js

```
import React from 'react';

const ChildComponent = (props) => {
  return (
    <div>
      <h3>Child Component</h3>
      <p>{props.message}</p>
    </div>
  );
};

export default ChildComponent;
```

ParentComponent.js

```
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  const data = 'Hello from Parent!';

  return (
    <div>
      <h2>Parent Component</h2>
      <ChildComponent message={data} />
    </div>
  );
};

export default ParentComponent;
```

App.js

```
import React from 'react';
import ParentComponent from './ParentComponent';
import './App.css';

function App() {
  return (
    <div className="App">
      <ParentComponent />
    </div>
  );
}

export default App;
```

7. **Create a React application that uses Props in Class based Components**

```
npx create-react-app props-class-based-example
cd props-class-based-example
```

ChildComponent.js

```
import React from 'react';

function ChildComponent(props) {
  const { name, age, bio } = props.person;

  return (
    <div>
      <h2>Child Component</h2>
      <p>Name: {name}</p>
      <p>Age: {age}</p>
      <p>Bio: {bio}</p>
    </div>
  );
}

export default ChildComponent;
```

ParentComponent.js

```
import React, { Component } from 'react';
import './App.css';
import ChildComponent from './ChildComponent';

class ParentComponent extends Component {
  render() {
    const person = {
      name: "John Doe",
      age: 30,
      bio: "A software developer",
    };

    return (
      <div className="App">
        <h1>Parent Component</h1>
        <ChildComponent person={person} />
      </div>
    );
```

```
  }
}

export default ParentComponent;
```

App.js

```
import React from 'react';
import ParentComponent from './ParentComponent';
import './App.css';

function App() {
  return (
    <div className="App">
      <ParentComponent />
    </div>
  );
}

export default App;
```

8. Create simple react application using state and class based component

```
npx create-react-app state-class-based-example
cd state-class-based-example
```

App.js

```
import React, { Component } from 'react';
import './App.css';

class CounterApp extends Component {
  constructor(props) {
    super(props);

    // Initialize the state with a counter value of 0
    this.state = {
      counter: 0,
    };
  }

  // Function to increment the counter
  incrementCounter = () => {
    this.setState({ counter: this.state.counter + 1 });
  };

  // Function to decrement the counter
  decrementCounter = () => {
    this.setState({ counter: this.state.counter - 1 });
  };

  render() {
    return (
      <div className="App">
        <h1>Counter App</h1>
        <p>Counter: {this.state.counter}</p>
        <button onClick={this.incrementCounter}>Increment</button>
        <button onClick={this.decrementCounter}>Decrement</button>
      </div>
    );
  }
}

export default CounterApp;
```

9. **Create a React application that uses Props in Functional Components and useState Hook**

```
npx create-react-app props-functional-components-example
cd props-functional-components-example
```

App.js

```
import React, { useState } from 'react';
import './App.css';

function Person(props) {
  const [name, setName] = useState(props.initialName);

  const handleNameChange = (newName) => {
    setName(newName);
  };

  return (
    <div>
      <h1>Person Component</h1>
      <p>Name: {name}</p>
      <button onClick={() => handleNameChange('John')}>Change Name to
John</button>
      <button onClick={() => handleNameChange('Jane')}>Change Name to
Jane</button>
    </div>
  );
}

function App() {
  return (
    <div className="App">
      <h1>My React App</h1>
      <Person initialName="Alice" />
    </div>
  );
}

export default App;
```

10. Create a react application to create a form component and handle form submissions and user input using state.

```
npx create-react-app form-component-example
cd form-component-example
```

App.js

```
import React, { useState } from 'react';
import './App.css';

function FormComponent() {
 const [formData, setFormData] = useState({
  name: '',
  email: '',
 });

 const handleInputChange = (e) => {
  const { name, value } = e.target;
  setFormData({
   ...formData,
   [name]: value,
  });
 };

 const handleSubmit = (e) => {
  e.preventDefault();
  alert(`Name: ${formData.name}\nEmail: ${formData.email}`);
 };

 return (
  <div>
   <h2>Form Component</h2>
   <form onSubmit={handleSubmit}>
    <div>
     <label htmlFor="name">Name:</label>
     <input
      type="text"
      id="name"
      name="name"
      value={formData.name}
      onChange={handleInputChange}
     />
    </div>
    <div>
     <label htmlFor="email">Email:</label>
     <input
      type="email"
```

```
          id="email"
          name="email"
          value={formData.email}
          onChange={handleInputChange}
        />
      </div>
      <button type="submit">Submit</button>
    </form>
   </div>
  );
}

function App() {
  return (
   <div className="App">
    <h1>React Form Component Example</h1>
    <FormComponent />
   </div>
  );
}

export default App;
```

11. Rendering Data:
    - Render a list of items using `map`.
    - Display dynamic data in your components.
12. Event Handling:
    - Add event handlers to components.
    - Update state based on user interactions.
13. Conditional Rendering:
    - Conditionally render components or elements.
    - Implement conditional rendering based on user input.

14. Component Lifecycle:
    - Understand the lifecycle methods in class-based components (e.g., `componentDidMount`, `componentDidUpdate`, etc.).
    - Use `useEffect` for lifecycle management in functional components.
15. Routing:
    - Implement basic routing using `react-router-dom`.
    - Create routes and navigate between different views/pages.
16. API Integration:
    - Fetch data from a public API (e.g., JSONPlaceholder) and display it in your React app.

17. State Management:
    - Integrate Redux or the Context API for global state management.
    - Implement actions and reducers for state updates.
18. Context API:
    - Explore and practice using the Context API for state management.
19. Hooks:
    - Learn and use various React hooks like `useEffect`, `useContext`, `useRef`, etc., in your components.