WHAT IS JSX?

JSX (JavaScript XML) is a syntax extension is used in to write JavaScript code with XML-like syntax for defining the structure and content of UI components.

JSX Example

1.

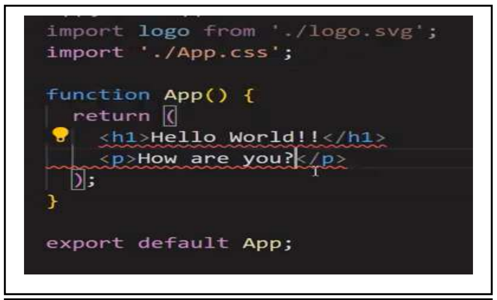
```
function Greeting(props) {
  return (
<div>
<h1>Welcome, {props.name}!</h1>
<p>Today is {props.day}.</p>
</div>
  );
}
export default Greeting;
```

```
import Greeting from './Greeting';
function App() {
  return (
<div>
<Greeting name="NAINCY" day="Monday"
/>
</div>
  );
}
export default App;
```

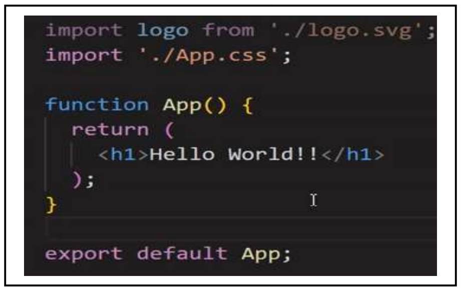
React: At the top of your file, you need to import the React library to use JSX. You can do this with the following line of code:

1. ***import React from 'react';***

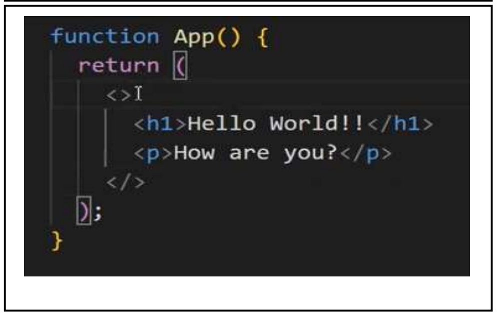
2. **Single Root Element:** *JSX requires that you have a single root element in your component. This means that all JSX code must have only one top-level element. If you need multiple elements, you can wrap them in a <div> or use a Fragment (<React.Fragment> or shorthand <>) to avoid adding an unnecessary extra wrapper element.*



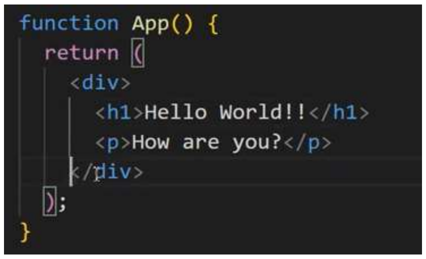
```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <h1>Hello World!!</h1>
    <p>How are you?</p>
  );
}

export default App;
```



```
import logo from './logo.svg';
import './App.css';

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

export default App;
```



```
function App() {
  return (
    <>
      <h1>Hello World!!</h1>
      <p>How are you?</p>
    </>
  );
}
```



```
function App() {
  return (
    <div>
      <h1>Hello World!!</h1>
      <p>How are you?</p>
    </div>
  );
}
```

3. ***Self-closing Tags**: If a JSX tag doesn't have any children, you can use a self-closing syntax:*
    2. **<img src="image.jpg" alt="An image" />**

4. **JavaScript Expressions:** JSX allows you to embed JavaScript expressions within curly braces {}. You can use variables, functions, or any valid JavaScript expression inside the curly braces:
    const name = 'John';
    <h1>Hello, {name}!</h1>

3. ***class vs. ClassName:*** JSX uses className instead of class for specifying CSS classes. This is because class is a reserved keyword in JavaScript:
   <div className="my-class"></div>

4. ***Inline Styles:*** To apply inline styles in JSX, you use a JavaScript object instead of a string:
```
const style = {
  color: 'red',
fontSize: '16px'
};
<div style={style}>Some text</div>
```
5. ***Event Handling:*** You can attach event handlers to JSX elements using camel-cased event names. The event handler should be a function or a method defined in the component
```
function handleClick() {
console.log('Button clicked!');
}
<button onClick={handleClick}>Click Me</button>
```

6. ***Conditional Rendering:*** You can use JavaScript's conditional statements like if and the ternary operator ? inside JSX to conditionally render elements:

```
const isLoggedIn = true;
<div>
  {isLoggedIn ?<p>Welcome, user!</p> : <p>Please log in.</p>}
</div>
```

7. **Comments:** JSX supports JavaScript-style comments within curly braces {/* */} or within the rendered output using // or /* */:
```
{/* This is a JSX comment */}
<div>
          {/* This comment won't be rendered */}
        <p>Hello, world!</p>
        </div>
```

8. ***Components:*** JSX allows you to define and use custom components. Custom components are written as capitalized names and can be used like HTML tags:
   ```
   function MyComponent() {
     return <div>Hello, I am a custom component!</div>;
   }
   // Usage:
   <MyComponent />
   ```

9. **Props:** You can pass data to custom components using props (short for properties). Props are similar to HTML attributes and are accessed within the component as properties of the props object:

   ```
   function Greeting(props) {
     return <h1>Hello, {props.name}!</h1>;
   }
   // Usage:
   <Greeting name="John" />
   ```

10. ***Fragments:*** Fragments allow you to group a list of elements without adding an extra wrapper element to the DOM. You can use the <React.Fragment> component or the shorthand <> and </> syntax:
    ```
    function MyComponent() {
      return (
    <>
    <h1>Title</h1>
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
    </>
      );
    }
    ```

11. ***HTML Entities:*** JSX does not support HTML entities like &nbsp; or &copy; directly. Instead, you can use Unicode characters or escape sequences:
    e.g.
    ```
    <div>&#169;</div>
    // Or
    <div>&copy;</div>
    // Or
    <div>&#x00A9;</div>

    function MyComponent() {
      return <div>&copy;The Target </div>;
    }
    ```

Note:

To use JSX in your projects, you need a transpiler like Babel. Babel can transform JSX code into regular JavaScript code that browsers can understand.