Components and Propertties

Components

It defines the **visuals and interactions** that make up what people see in app. Declares **how the view looks like, given the data**. When the data changes, if you are used to the jQuery way of doing things, you'd typically do some DOM manipulation. But the React doesn't do anything like that! The React library figures out how the new view looks, and just applies the changes between the old view and the new view. This makes the views consistent, predictable, easier to maintain, and simpler to understand.

Why Components?

There may be elements which are similar . There might be a need to make a change that will result in changes in multiple places. Similar to functions, if we could write code related to the element in one place, changes can be minimized. Solution is to have reusable piece of JavaScript code that output (via JSX) HTML elements

Types of Components

1. **Stateless Functional Component –** [Will be discussed later]
    - Includes simple JavaScript functions and immutable properties, i.e., the value for properties cannot be changed.
    - Use hooks to achieve functionality for making changes in properties using JS.
    - Used mainly for UI.

2**. Stateful Class Component**
    - Classes which extend the Component class from React library. The class component must include the render method which returns HTML.

First Stateful component

Creation of a component:

```
class HelloWorld extends React.Component
{       render()
        {   return <p>Hello, componentized world!</p>; }
}
```

The render() method is something that that the React calls when it needs to display the component. There are other methods with special meaning to React that can be implemented, called the Lifecycle functions, which provide hooks into various stages of the component formation and events. This will be discussed later. But render() is one that must be present; otherwise you'll have a component that has no screen presence.

Calling/Rendering a component:

Case 1: Add the JSX in the render method with a element with the tag name as the Component name

```
ReactDOM.render(  <HelloWorld/>,
        document.querySelector("#container")
);
```

Case 2: Add the NON-JSX in the render method with a element using React.createElement

```
ReactDOM.render( React.createElement("HelloWorld",null, null),
        document.querySelector("#container")
);
```

Parameterized components:

Passing the attributes to the components during rendering and using these attributes as properties inside the component.

Properties

Properties are ways in which React components can be customized. Immutable and same as what attributes in HTML elements. Props are arguments passed into React components and are passed via HTML attributes. 2 steps to add properties to components

- Make the function of your component read the props from the props parameter Place the props inside curly brackets – { }. In JSX, if you want something to get evaluated as an expression, you need to wrap that something inside curly brackets. If you don't do that, the raw text gets printed out.

```
return <h1>Hello, {this.props.greetingtarget}</h1>
```

- **Modify the Component Call** : When rendering the component, add the prop to the component using the attribute

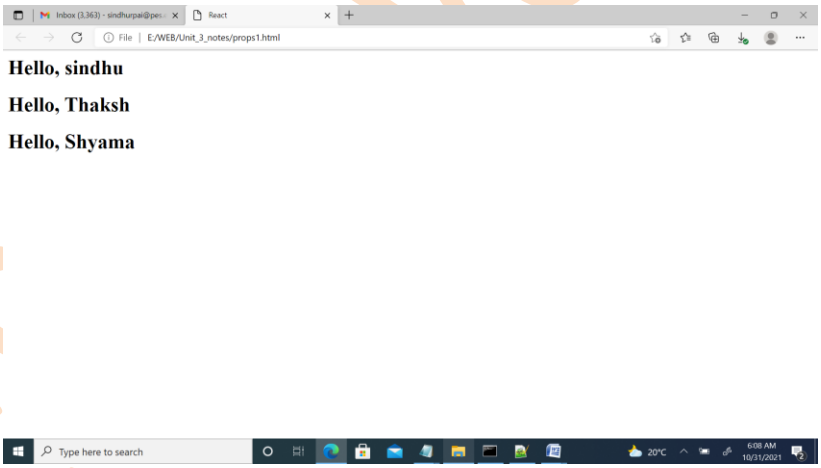
```
<Helloworld greetingtarget = "sindhu" />
```

Coding example 1: Usage of this.props

```
<html>
    <head>
        <title>React</title>
        <script crossdomain
src="https://unpkg.com/react@16/umd/react.development.js"> </script>
        <script crossdomain src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"> </script>
        <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js">
</script>
    </head>
    <body>
        <div id="container"></div>
        <script type="text/babel">
            var dest = document.querySelector("#container");
            class Helloworld extends React.Component {
                render(){
                    return <h1>Hello, {this.props.greetingtarget}</h1>
                    }
```

```
            }
            ReactDOM.render(
                <div>
                    <Helloworld greetingtarget = "sindhu" />
                    <Helloworld greetingtarget = "Thaksh" />
                    <Helloworld greetingtarget = "Shyama" />
                </div>,
                document.getElementById("container")
            );
        </script>
    </body>
</html>
```

Output:



Usage of this.props.children

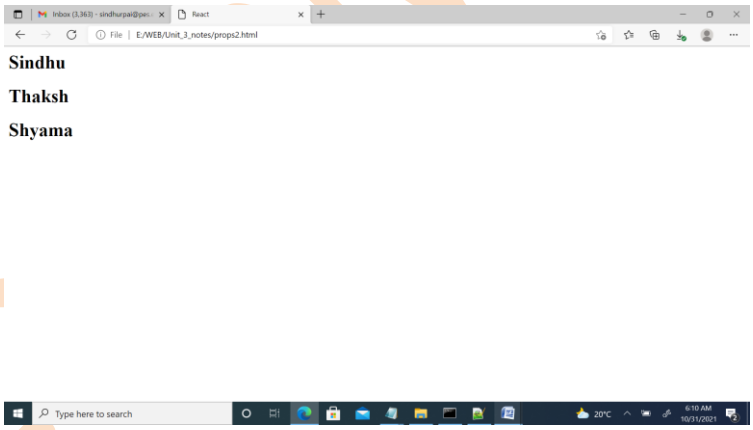
A special property that is passed to components automatically. It is used to display the data between the opening and closing JSX tags when invoking a component. Can have one element, multiple elements, or none at all. It's value will be respectively a single child node, an array of child nodes or undefined

Coding example 2:

```
<script type="text/babel">
```

```
class Helloworld extends React.Component {
        render(){
                return  <h1>{this.props.children}</h1>
                }
    }
ReactDOM.render(
        <div>
                <Helloworld>Sindhu</Helloworld>
                <Helloworld>Thaksh</Helloworld>
                <Helloworld>Shyama</Helloworld>
        </div>,
        document.querySelector("#container")
);
</script>
```

Output:



Validating the property values

The properties being passed to component to can be validated against a specification. This specification is supplied in the form of a static object called propTypes in the class, with the name of the property as the key and the validator as the value, which is one of the many constants exported by React.PropTypes, for example,

React.PropTypes.string

```
Helloworld.propTypes = { greetingtarget: React.PropTypes.string.isRequired,
                         greetingtarget_id: React.PropTypes.number
                       };
```

Property validation is checked only in development mode, and a warning is shown in the console when any validation fails. Since we are in an early stage in the development of the application, we expect more changes to the properties. You can also default the property values when the parent does not supply the value. For example, if you want the greeting_target to be defaulted to something else, rather than show an empty string, you can do this: `Helloworld.defaultProps = { greetingtarget: '-- no title --', };`

Few Points to think:

- Can we have nested components?
- Can we provide style to components?
- How to send properties from one component to another?
- What are the different methods under life cycle of components?

  Refer to the further notes for more details on the above points