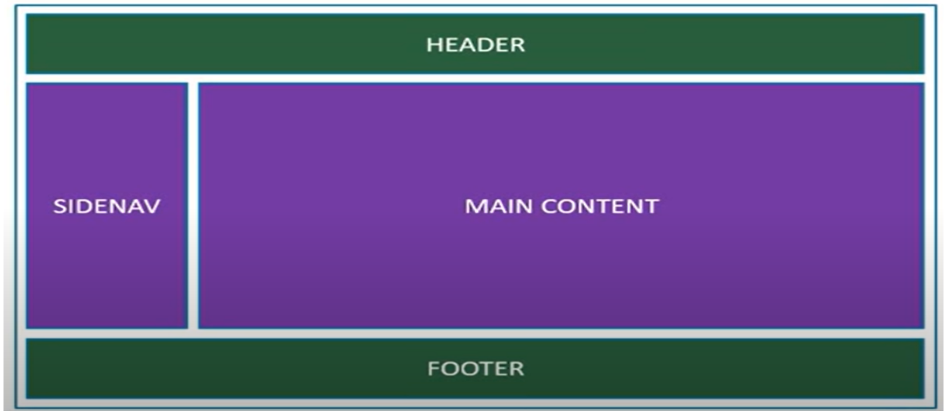
REACT JS

Introduction :

1. It is a Open Source Library for Building UI
2. It completely focuses on UI it will not cater to routing and Http Requests unlike Angular
3. It requires other libraries support to cater routing and Http Requests

Let's React :

1. It has Component based Architecture
2. 

3. React is Reusable and Declarative since DOM (Document Object Model) updates are handled gracefully by react.
4. React can be integrated easily into other applications.

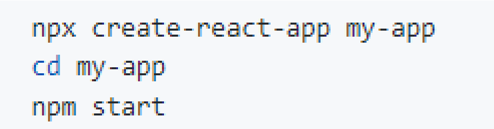
PreRequisites:

1. Installation of Node JS, VS Code
2. Basic Html, CSS and JavaScript with proficiency in 'this', filter,map,reduce
3. ES6(Typescript) - let & const, arrow functions, template & object literals, rest and separate operators, default parameters, destructuring assignment
4. Version used : **16.5.2**

Flow of Learning:

❖ Fundamentals — Http — Routing — Redux — Utilities

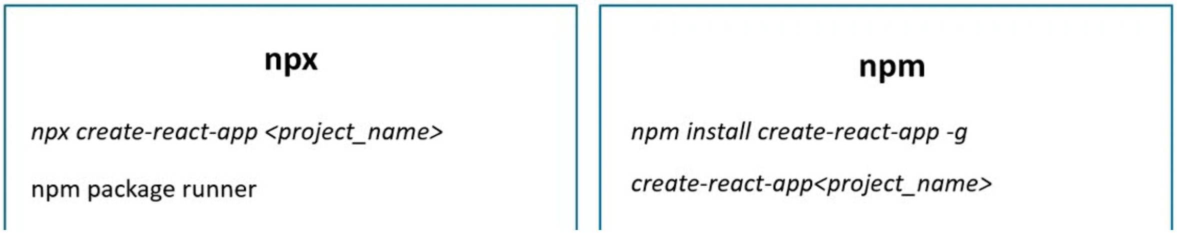
Commands :

1. 

```
npx create-react-app my-app
cd my-app
npm start
```

2. These commands will create a react project (my-app), move into that workspace and start compiling the react project
3. Default React Port : http://localhost:3000

Create React App :



npx

npx create-react-app <project_name>

npm package runner

npm

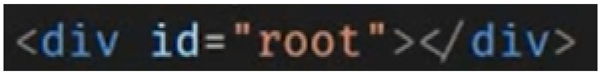
npm install create-react-app -g

create-react-app<project_name>

- npm creates the project globally hence npx is preferred compared to npm

Folder Structure :

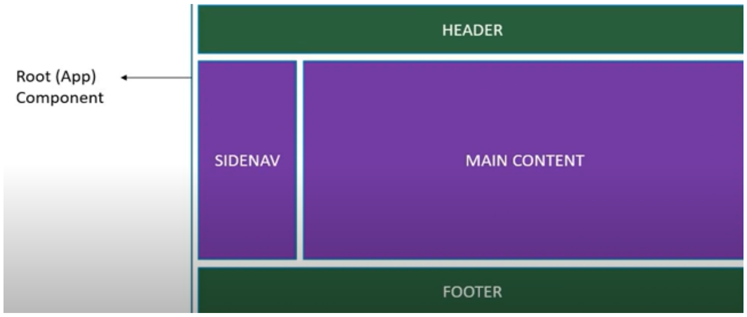
1. Root Level : 3 folders and 4 files
   a. Folders : **node_modules, public, src**
   b. Files : **.gitignore, package-lock.json, package.json, README.md**
2. **package.json** file consists of dependencies and scripts required for the application
3. **public** folder contains 3 files
   a. Files: **favicon.ico, index.html, manifest.json**

4. **index.html** file is the only html file in the entire application
    a. It consists of a div tag used by react to control the UI
    b. 

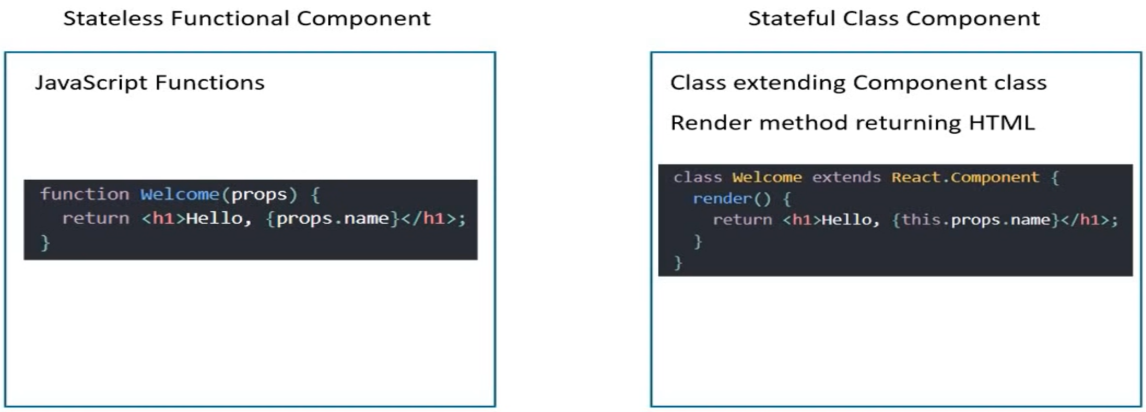
```
<div id="root"></div>
```

5. **src** folder contains 7 files
    a. Files: **App.css**, **App.js, App.test.js, index.css**, **index.js**, logo.svg, **serviceWorker.js**
    b. **index.js** is the starting point of the application which redirects us to start working from **App.js**

Components:

1. Components describe a part of UI (Facebook has over 30k components)
2. 

3. They are reusable and can be nested inside other components.
4. Components are the building blocks of any react application.
5. Component code is generally placed in a **.js** file
6. Component Types :
    a. Stateless Functional Component
    b. Stateful Class Component (Ex : **App.js**)

7. 

Stateless Functional Component

JavaScript Functions

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

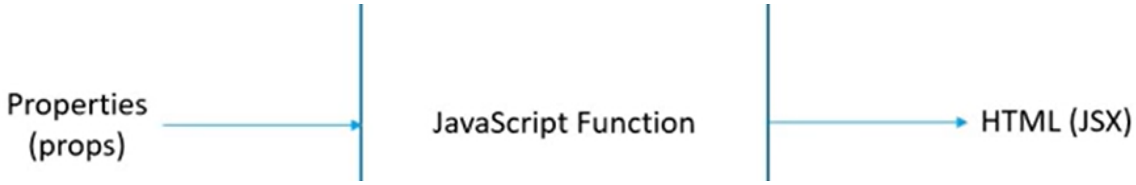
Stateful Class Component

Class extending Component class

Render method returning HTML

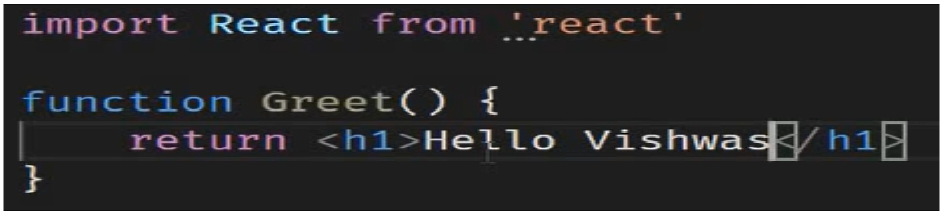
```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Functional Components:

1. 

Properties (props) → JavaScript Function → HTML (JSX)

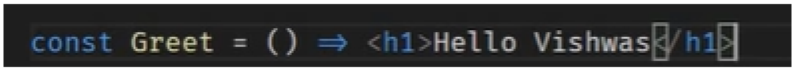
2. Naming Convention used for naming files : **PascalCase**
   a. PascalCase is a programming naming convention where the first letter of each compound word in a variable is capitalized.

3. Creating custom component : **Greet.js**
   a. 

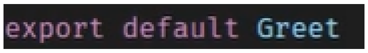
```
import React from 'react'

function Greet() {
    return <h1>Hello Vishwas</h1>
}
```

   b. (or) we can use a simple arrow function



```
const Greet = () => <h1>Hello Vishwas</h1>
```

4. Integrating this custom component (**Greet.js**) into main component (**App.js**)
   a. Add this line in every custom component: 

```
export default Greet
```

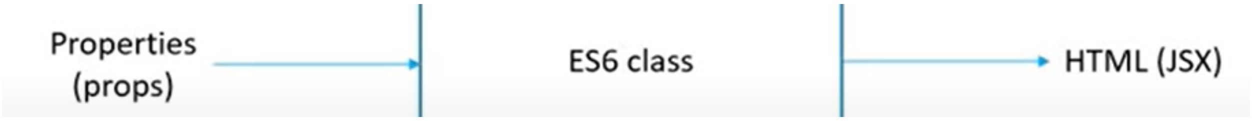
b. Using this line you can import this component in main component with the desired name

c. Importing the custom component in **App.js**

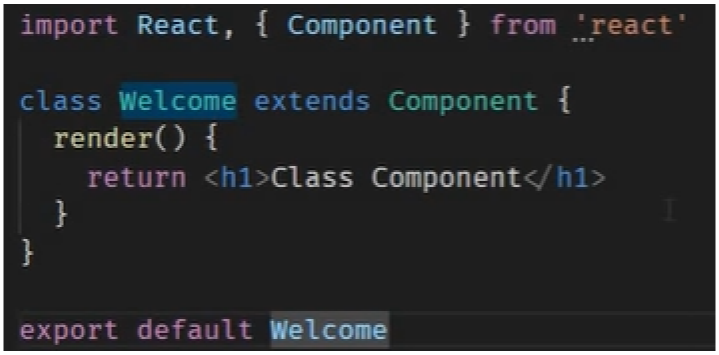
d. adding **<Greet></Greet>** or **<Greet />** in html area of **App.js**

Class Components:

1.



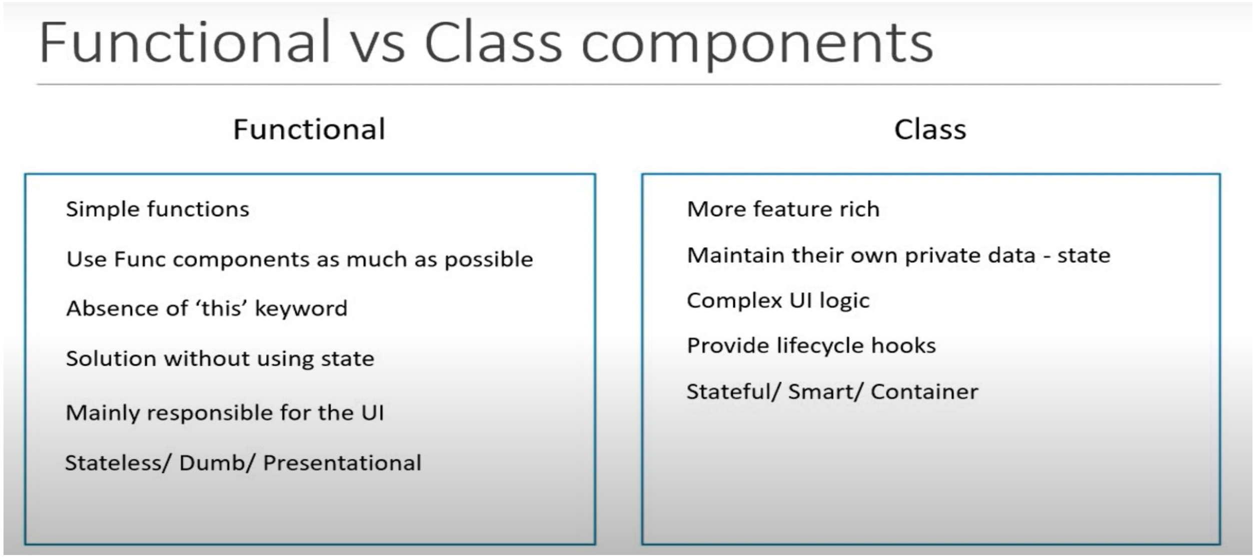
2.



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

class Welcome extends Component {
  render() {
    return <h1>Class Component</h1>
  }
}

export default Welcome
```



Functional vs Class components

Functional

Simple functions

Use Func components as much as possible

Absence of 'this' keyword

Solution without using state

Mainly responsible for the UI

Stateless/ Dumb/ Presentational

Class

More feature rich

Maintain their own private data - state

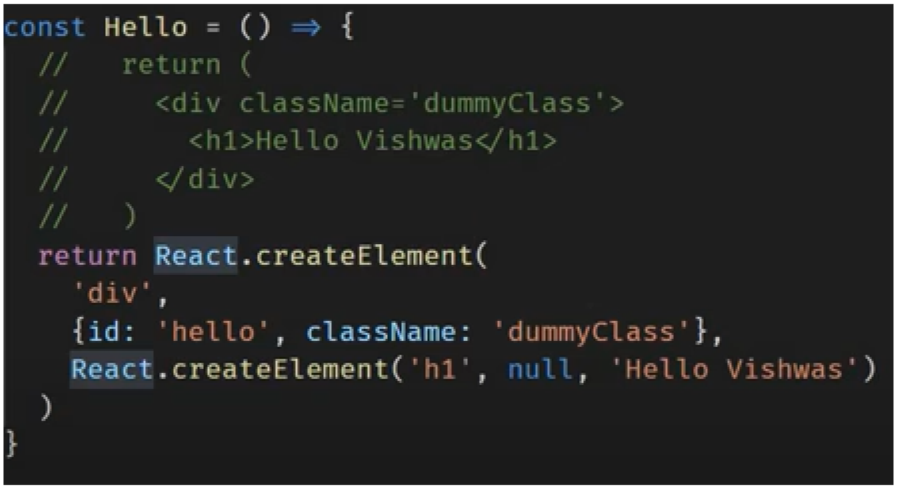
Complex UI logic

Provide lifecycle hooks

Stateful/ Smart/ Container

- Correction:
    - Lifecycle hooks Exists both in Functional and class components

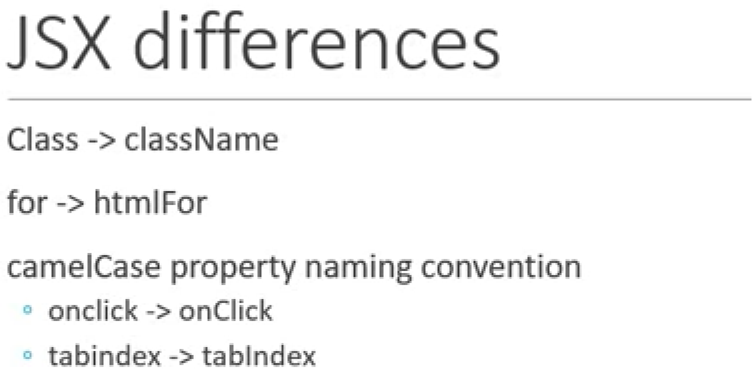
JSX:

1. JavaScript XML (JSX) : Extension to the JavaScript language syntax
2. JSX tags have a tag name, attributes and children
3. JSX makes react code simpler and elegant
4. 

```
const Hello = () => {
  //   return (
  //     <div className='dummyClass'>
  //       <h1>Hello Vishwas</h1>
  //     </div>
  //   )
  return React.createElement(
    'div',
    {id: 'hello', className: 'dummyClass'},
    React.createElement('h1', null, 'Hello Vishwas')
  )
}
```

   a. With & without JSX
   b. Commented part is using JSX and uncommented is without JSX

5. JSX differences with html

   a. 

JSX differences

Class -> className

for -> htmlFor

camelCase property naming convention
- onclick -> onClick
- tabindex -> tabIndex