Functional Component

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML. Components come in two types, Class components and Function components, in this tutorial we will concentrate on Function components.

<article>

<h1>My First Component</h1>

<ol>

<li>Components: UI Building Blocks</li>

<li>Defining a Component</li>

<li>Using a Component</li>

</ol>

</article>

This markup represents this article <article>, its heading <h1>, and an (abbreviated) table of contents as an ordered list <ol>. Markup like this, combined with CSS for style, and JavaScript for interactivity, lies behind every sidebar, avatar, modal, dropdown—every piece of UI you see on the Web.

React lets you combine your markup, CSS, and JavaScript into custom “components”, reusable UI elements for your app. The table of contents code you saw above could be turned into a <TableOfContents /> component you could render on every page. Under the hood, it still uses the same HTML tags like <article>, <h1>, etc.

Just like with HTML tags, you can compose, order and nest components to design whole pages. For example, the documentation page you’re reading is made out of React components:

<PageLayout>

<NavigationHeader>

<SearchBar />

<Link to="/docs">Docs</Link>

</NavigationHeader>

<Sidebar />

<PageContent>

<TableOfContents />

<DocumentationText />

</PageContent>

</PageLayout>

**Creating React Component**

**Functional Component**

**React components are regular JavaScript functions, but their names must start with a capital letter or they won’t work!**

export default function Profile() {

return (

<img

src="https://i.imgur.com/MK3eW3Am.jpg"

alt="Katherine Johnson"

/>

)

}

function Profile() {

return (

<img

src="https://i.imgur.com/MK3eW3As.jpg"

alt="Katherine Johnson"

/>

);

}

export default function Gallery() {

return (

<section>

<h1>Amazing scientists</h1>

<Profile />

<Profile />

<Profile />

}

Using a JavaScript function, we can make a functional React component.

// React component syntax

// it can be arrow function, function declaration or function expression

const jsx = <tag> Content </tag>

const ComponentName = () => {

return jsx

}

The following expression is a JSX element.

// JSX element, header

const header = (

<header style={headerStyles}>

<div className='header-wrapper'>

<h1>Welcome to 30 Days Of React</h1>

<h2>Getting Started React</h2>

<h3>JavaScript Library</h3>

<p>Asabeneh Yetayeh</p>

<small>Oct 3, 2020</small>

</div>

</header>

)

// React Component

const Header = () => {

return header

}

// or we can just return the JSX

const Header = () => {

return (

<header style={headerStyles}>

<div className='header-wrapper'>

<h1>Welcome to 30 Days Of React</h1>

<h2>Getting Started React</h2>

<h3>JavaScript Library</h3>

<p>Asabeneh Yetayeh</p>

<small>Oct 3, 2020</small>

</div>

</header>

)

}

// Even th above code can be written like this

// Explicitly returning the JSX

const Header = () => (

<header style={headerStyles}>

<div className='header-wrapper'>

<h1>Welcome to 30 Days Of React</h1>

<h2>Getting Started React</h2>

<h3>JavaScript Library</h3>

<p>Asabeneh Yetayeh</p>

<small>Oct 3, 2020</small>

</div>

</header>

)

**Rendering components**

Now, lets change all the JSX elements we had to components. When we call JSX element we use curly brackets and when we call components we do as follows . If we pass an attribute, when we call the component name, we call it props(<ComponentName propsName = {'data-type'} />). We will talk about props in another section.Live on code pen

Let's render first the Header component.

// index.js

import React from 'react'

import ReactDOM from 'react-dom'

// Header Component

const Header = () => (

<header>

<div className='header-wrapper'>

<h1>Welcome to 30 Days Of React</h1>

<h2>Getting Started React</h2>

<h3>JavaScript Library</h3>

<p>Asabeneh Yetayeh</p>

<small>Oct 3, 2020</small>

</div>

</header>

)

const rootElement = document.getElementById('root')

// we render the JSX element using the ReactDOM package

ReactDOM.render(<Header />, rootElement)

Now, let's create an App component , that will wrap the Header, Main and Footer. Then the App component will be render on the DOM.

// index.js

import React from 'react'

import ReactDOM from 'react-dom'

import asabenehImage from './images/asabeneh.jpg'

// Header Component

const Header = () => (

<header>

<div className='header-wrapper'>

<h1>Welcome to 30 Days Of React</h1>

<h2>Getting Started React</h2>

<h3>JavaScript Library</h3>

<p>Asabeneh Yetayeh</p>

<small>Oct 3, 2020</small>

</div>

</header>

)

// User Card Component

const UserCard = () => (

<div className='user-card'>

<img src={asabenehImage} alt='asabeneh image' />

<h2>Asabeneh Yetayeh</h2>

</div>

)

// TechList Component

const TechList = () => {

const techs = ['HTML', 'CSS', 'JavaScript']

const techsFormatted = techs.map((tech) => <li key={tech}>{tech}</li>)

return techsFormatted

}

// Main Component

const Main = () => (

<main>

<div className='main-wrapper'>

<p>Prerequisite to get started react.js:</p>

<ul>

<TechList />

</ul>

<UserCard />

</div>

</main>

)

// Footer Component

const Footer = () => (

<footer>

<div className='footer-wrapper'>

<p>Copyright 2020</p>

</div>

</footer>

)

// The App, or the parent or the container component

const App = () => (

<div className='app'>

<Header />

<Main />

<Footer />

</div>

)

const rootElement = document.getElementById('root')

// we render the App component using the ReactDOM package

ReactDOM.render(<App />, rootElement)

**Rendering Components**

**Injecting data to JSX in React Component**

So far, we used static data on the JSX elements. Now let's pass different data types as dynamic data. The dynamic data could be strings, numbers, booleans, arrays or objects. Let us see each of the data types step by step. To inject data to a JSX we use the {} bracket.

In this section we inject only strings

import React from 'react'

import ReactDOM from 'react-dom'

const welcome = 'Welcome to 30 Days Of React'

const title = 'Getting Started React'

const subtitle = 'JavaScript Library'

const firstName = 'Asabeneh'

const lastName = 'Yetayeh'

const date = 'Oct 3, 2020'

// JSX element, header

const header = () => {

return (

<header>

<div className='header-wrapper'>

<h1>{welcome}</h1>

<h2>{title}</h2>

<h3>{subtitle}</h3>

<p>

Instructor: {firstName} {lastName}

</p>

<small>Date: {date}</small>

</div>

</header>

)

}

const rootElement = document.getElementById('root')

// we render the App component using the ReactDOM package

ReactDOM.render(<Header />, rootElement)

Similar to the Header component we can implement to Main and Footer component.

// To get the root element from the HTML document

const rootElement = document.querySelector('.root')

// JSX element, header

const welcome = 'Welcome to 30 Days Of React Challenge'

const title = 'Getting Started React'

const subtitle = 'JavaScript Library'

const author = {

firstName: 'Asabeneh',

lastName: 'Yetayeh',

}

const date = 'Oct 2, 2020'

// JSX element, header

const Header = () => (

<header>

<div className='header-wrapper'>

<h1>{welcome}</h1>

<h2>{title}</h2>

<h3>{subtitle}</h3>

<p>

Instructor: {author.firstName} {author.lastName}

</p>

<small>Date: {date}</small>

</div>

</header>

)

const numOne = 3

const numTwo = 2

const result = (

<p>

{numOne} + {numTwo} = {numOne + numTwo}

</p>

)

const yearBorn = 1820

const currentYear = 2020

const age = currentYear - yearBorn

const personAge = (

<p>

{' '}

{author.firstName} {author.lastName} is {age} years old

</p>

)

// User Card Component

const UserCard = () => (

<div className='user-card'>

<img src={asabenehImage} alt='asabeneh image' />

<h2>

{author.firstName} {author.lastName}

</h2>

</div>

)

// JSX element, main

const techs = ['HTML', 'CSS', 'JavaScript']

const techsFormatted = techs.map((tech) => <li key={tech}>{tech}</li>)

// JSX element, main

const Main = () => (

<main>

<div className='main-wrapper'>

<div>

<p>

Prerequisite to get started{' '}

<strong>

<em>react.js</em>

</strong>

:

</p>

<ul>{techsFormatted}</ul>

{result}

{personAge}

</div>

<UserCard />

</div>

</main>

)

const copyRight = '2020'

// JSX element, footer

const Footer = () => (

<footer>

<div className='footer-wrapper'>

<p>Copyright &copy;{copyRight}</p>

</div>

</footer>

)

// JSX element, app

const app = () => (

<div className='app'>

<Header />

<Main />

<Footer />

</div>

)

// we render the App component using the ReactDOM package

ReactDOM.render(<App />, rootElement)

Further on Functional components

We have transformed all the JSX elements of Day 2 to functional components, and by now you are very familiar with components. Let's create more components. What is the smallest size of a component? A component that returns only a single HTML as JSX is considered as a small component. A button component or an alert box component, or just an input field component.

const Button = () => <button>action</button>

The Button component is made of a single HTML button element. Let's style this button using JavaScript style object. All CSS properties should be camelCase to make a JavaScript CSS object. If we pass a number without unit as CSS value, it is considered as px. See the example below.

const buttonStyles = {

padding: '10px 20px',

background: 'rgb(0, 255, 0',

border: 'none',

borderRadius: 5,

}

const Button = () => <button style={buttonStyles}> action </button>

The Button component is a dumb component, because it does not take any parameters and we cannot change the action text dynamically. We need to pass props to the button, to change the value dynamically. We will see props in the next section. Before we close today's lesson let's make another, more functional component, which displays a random hexadecimal number.

import React from 'react'

import ReactDOM from 'react-dom'

// Hexadecimal color generator

const hexaColor = () => {

let str = '0123456789abcdef'

let color = ''

for (let i = 0; i < 6; i++) {

let index = Math.floor(Math.random() \* str.length)

color += str[index]

}

return '#' + color

}

const HexaColor = () => <div>{hexaColor()}</div>

const rootElement = document.getElementById('root')

// we render the App component using the ReactDOM package

ReactDOM.render(<HexaColor />, rootElement)