React app made of components, JS functions that return markup with logic and appearance 🡪 <Component />

use camelCase as default

declare with export default function myApp() { return ({ XHTML code }); }

for non-default named import, import {Component} from ‘file’

can use conditional rendering on components

either conditional statements or {cond && ‘component/thing to render’}

import { StrictMode } from 'react'; for React

import { createRoot } from 'react-dom/client'; for React lib to talk with browser

Components: JS functions that return markup with logic and appearance 🡪 <Component />

can’t return multiple JSX tags, must have wrapper 🡪 use <Fragment> </Fragment = <> … </> if no semantic wrapper

for closing tags, send props.children: function Component ({ children }) 🡪 <Component> … </Component>

for attributes, use className instead of class, use {…} for vals from code

Props = data passed down to child components

declare shared states in parent component

use spread operator (…props) to send all props

for lists, each item has key prop for id

As pure functions, does not change objs/vars that existed before it was called and given same inputs, always same outputs

State: Component’s memory

updates only on next render, not during current render

treat vars/objects as immutable

when updating objects/arrays, make new then update object

local mutation is allowed to change an object, then use object in setState()

for event handling, create event handler in component 🡪 onEvent = handleEvent

Hooks = functions starting with useX that can only be used at the top of components

to update screen, import { useState } from 'react'; 🡪 const [var, setVar] = useState(initVal);

use null for an initial empty value

updater function queues series of state changes, use setState(v => {v + 1});

setState(v => v + 1) == setState(1)

function arg usually named first letters of state var

React Under-the-hood:

Creates a virtual DOM in memory to manipulate elements, then makes changes to browser DOM, instead of direct browser DOM manipulation

Changes only what needs to be changed

Need npm to use in production, which is included in Node.js

ES6: Standardized JS version published in 2015

Class = type of function, with properties assigned in constructor() method

Inheritance = extends keyword, used with super() to access parent’s attributes

Arrow Functions = shorthand function declaration

Destructuring

assign array items to vars => const [var1, var2, var3] = arr; where arr = [i1, i2, i3] 🡪 Can be used [var1,, var3] = arr

use object in func => func myFunc({prop1, prop2, prop3}) { …code… }

Spread Operator (…) = copy all/part of array/object into another

Ternary operator (? :) = shorthand if/else statement => cond ? true : false;

JSX = JavaScript XML, which can use { expression } for string interpolation

Must properly close HTML tags like in XML

Uses className instead of class attribute

Must use ternary operator inside expressions, can’t use if/else statements

Components are JS functions that return HTML code

Class Components = extends React.Component class to inherit its functions

class myComp extends React.Component { code }

Function Components = uses function instead of class to inherit from React.Component

function myComp() = { return JSX }

Components have attributes that can pass in properties to their JSX

If using class, must put props in constructor

state object stores properties and re-renders component when changed

lifecycle:

Mounting = placing elements into DOM (will always call render)

constructor() – comp is initiated, sets up initial state and values, use props as args

getDerivedStateFromProps() – before comp is rendered, sets state based on props, use state as arg

render() – outputs HTML to DOM

componentDidMount() – after comp is rendered, for code that needs comp mounted

Updating = update comp when state or props change (always calls render)

getDerivedStateFromProps() – before comp is rendered, sets state based on props, use state as arg

shouldComponentUpdate() – checks whether to continue with the render, defaults true

render() – outputs HTML to DOM

getSnapshotBeforeUpdate() – before component is updated, allows access to previous state and props

if included, then componentDidUpdate() is required

componentDidUpdate() – after comp update, for code that need comp updated

Unmounting = remove elements from DOM

componentWillUnmount() – before comp unmount, comp to be removed from DOM

props are read-only

events are camelCase, uses arrow functions

conditional rendering uses if/else statements, logical && operator, or ternary operator

Lists usually use map(), but items must have a key to allow render update properly

Array index is a last-resort key

Forms use various hooks, name for controls

textarea value in value attr, select default in value attr

Create-react-app does not include page routing, so need router library (usually react router)

Can have multiple <Routes> components

Use Link for internal path, <a> for external

memo makes React skip comp render if props have not changed

Hooks let functional comps use React features => Can only be called in function comps at the top level, and cant be conditional

=> conventionally starts with “use”, keep in mind for custom ones

useState – track state

accepts initial state, and returns current state and state updater function

use spread operator on objects to use fill in previous state

useEffect – allows side effects on components when rendered e.g. fetch data, update DOM, timers

accepts function and dependencies, 2nd is optional

no dependency = run on every render; empty array = run on first render; prop/state = run on first render and when value changes

for effect cleanup, include a return function at the end of the hook e.g. timeouts, subscriptions, event listeners, or anything no longer needed

useContext – manage state globally

use createContext, then wrap comp tree with Context Provider with state value

useRef – stores values between renders, usually mutable values that don’t cause re-renders; access DOM element directly

accepts initial value for current Object, returns current

use ref attribute on HTML element to access in DOM directly

useReducer – sets custom state logic, like useState

accepts reducer function and initial state, returns current state and dispatch method

useCallback – isolate resource-intensive functions to not auto-run on renders, runs only when dependency updates

returns a memo-ized callback function

due to referential equality, when component renders, its functions are recreated

useMemo – caches values so it doesn’t need to be recalculated, runs only when dependency updates

accepts a function and dependencies, returns a memo-ized value