1. Getting Started
   1. REACT – JS library used for building user interfaces
   2. Component Based
      1. Encapsulation of their own state
      2. Each component implements a *render()* method
         1. Takes in input data and returns what to display
         2. Input data that is passed into component can be accessed by *render()* via *this.props*
      3. Stateful Component
         1. Internal state data via *this.state*
         2. When state data changes then *render()*
2. TaniaRascia Tutorial
   1. *render()* – only required method in a class component
   2. In index.js import React, ReactDOM, and ./index.css
   3. JSX – Javascript XMl
      1. JSX ex – “const heading = <h1 className="site-heading">Hello, React</h1>”
      2. NO JSX ex – “const heading = React.createElement('h1', { className: 'site-heading' }, 'Hello, React!')”
      3. Key Differences
         1. Use className not class
         2. camelCase so onclick is onClick
         3. Self closing tags must end in a slash <img />
   4. Components – two types: class or simple
      1. Most components get their own file
      2. Class Components
         1. Must include render() and return can only return one parent element
      3. Simple Components
         1. A function does not use the class keyword
         2. ES6 version –

const SimpleComponent = () => {

return <div>Example</div>

}

* 1. Props – data that is passed through to a component
     1. Component cannot change the props read only
  2. State – any data that should be saved and modified
     1. Must use *this.setState()* to modify an array
     2. Filter() – creates a new array
  3. Submitting Form Data
  4. Pulling in API Data
  5. Building and Deploying a React App

1. Intro to React – reactjs.org
   1. Local Development Environment
      1. Npx
   2. Overview
      1. What is React?
         1. Components – takes in parameters called *props* and returns a hierarchy of views via *render()*
      2. Inspecting the Starter Code
      3. Passing Data Through Props
      4. Interactive Component
         1. State is used to remember things
         2. *this.state* is private and is in the constructor
         3. *this.setState* tells react to re render the square
   3. Completing the Game
      1. Lifting the State Up
         1. To collect data from multiple children OR have two child components communicate with each other you NEED to declare the shared state in their parent component
         2. Parent can pass the state back down to the children by using props
      2. Why Immutability is Important
         1. Two approaches to changing data
            1. Mutate the data by directly changing the data’s value
            2. Replace the data with a new copy that has the desired changes

*Object.assign{}*

* + - 1. Complex Features Become Simple
         1. Avoiding direct data mutation lets us keep the previous versions of the games history intact and reuse them later
      2. Detecting Changes
         1. Detecting changes in immutable objects is considerably easier
      3. Determining When to Re render in REACT
         1. Help build pure components in REACT
         2. *ShouldComponentUpdate()*
    1. Function Components
       1. Function Components – components that only contain a render method and DON’T have their own state
          1. Doesn’t extend React.Component, just write a function that takes props as input and returns what should be rendered
    2. Taking Turns
    3. Declaring a Winner
  1. Adding Time Travel
     1. Storing a History of Moves
        1. Going to store past squares array in another array called history
     2. Lifting State up Again
        1. Place history in Game Component
        2. Resulting in remove the squares state from Board Child component
     3. Showing the Past Moves
        1. Map() – creates a new array populated with the results of calling a provided function on every element in the calling array
     4. Picking a Key
        1. Assign proper keys whenever you build a dynamic lists
     5. Implementing Time Travel
     6. Wrapping It Up

1. Main Concepts – reactjs.org
   1. Introducing JSX
      1. Embedding Expressions in JSX
         1. Ex below declare a variable called name and then use it inside JSX by wrapping it in CURLY BRACES
            1. Const name = ‘Josh Perez’;
            2. Const element = <h1>Hello, {name}</h1>;
         2. Can put any valid JS expressions inside CURLY BRACES
         3. Recommend wrap in parentheses to avoid pitfalls of semicolon insertion
      2. JSX is an Expression Too
         1. Can use JSX inside if statements or for loops, assign to variables, accept it as arguments, and return it from functions
      3. Specify Attributes with JSX
         1. Use “” specify string literals as attributes
         2. Use {} to embed a JS expression in an attribute
         3. ONLY USE QUOTES OR CURLY BRACES NEVER BOTH
      4. Specifying Children with JSX
         1. If tag is empty then can close immediately with />
         2. Other wise </name> at the end
      5. JSX Prevents Injection Attacks
         1. Everything is converted to string before being rendered
      6. JSX Represents Objects
         1. Babel complies JSX to React.createElement() calls
         2. It creates an object called React Elements
   2. Rendering Elements
      1. React elements are plain objects and are cheap to create
      2. Rendering an Element into the DOM
         1. DOM – represents the document as nodes and objects that way programming languages can interact with the page
         2. React apps usually have a single root DOM node
         3. *const root = ReactDOM.createRoot(document.getElementById("root"));*
         4. *root.render(<Game />);*
      3. Updating the Rendered Element
         1. Only way to update the UI is to create a new element and pass it to root.render()
      4. React Only Updates What’s Necessary
         1. DOM compares the element and its children to the previous one
         2. Only applies the DOM updates necessary to bring DOM to desired state
   3. Components and Props
      1. Function and Class Components
         1. Function Components
            1. Accepts a single props object argument and returns a react element
         2. Class Components
            1. Can use ES6 class to define a component
      2. Rendering a Component
         1. When React sees an element of a user defined component it asses JSX attributes and children to this component as a single object props
         2. Always start component names with a capital letter
      3. Composing Components
      4. Extracting Components
      5. Props are Read Only
         1. **ALL REACT COMPONENTS MUST ACT LIKE PURE FUNCTIONS WITH RESPECT TO THEIR PROPS**
   4. State and Lifecycle
      1. Adding Local State to a Class
         1. Replace this.props to this.state
         2. Add a class constructor that assigns an initial state
         3. Remove prop from class call
      2. Adding Lifecycle Methods to a Class
         1. Mounting
            1. When a class is rendered the first time
            2. componentWillMount() – runs after the component output has been rendered to the DOM
         2. Unmounting
            1. When a class is removed
            2. componentWillUnmount()
      3. Using State Correctly
         1. DON’T modify State directly
            1. Always use this.setState
            2. Only use this.state = when in the constructor
         2. State Updates may be asynchronous
         3. State Updates are Merged
            1. Can update the variables independently
         4. The Data Flows Down
            1. A component may choose to pass its state down as props to its child components (top-down)
   5. React.Component
      1. Must defined a render()
      2. Mounting – when an instance of a component is being created and inserted into DOM following in order
         1. Constructor()
            1. Need super(props)
            2. Only needed if making a state or binding methods
            3. NOT CALL setState in constructor (componentDidMount)
            4. Avoid copying props into state

Only use if intentionally want to ignore prop updates

* + - 1. Static getDerivedStateFromProps()
         1. Right before calling render()
         2. Both on initial mount and subsequent updates
         3. Return an object to udate the state or null to update nothing
         4. VERY RARE
      2. Render()
         1. Should examine this.props and this.state ad return one of the following types:

React Elements

JSX

Arrays and Fragments

Portals

Strings and Numbers

Booleans or nulls

* + - * 1. Should be pure, doesn’t modify the component state
        2. Will not be called if should ComponentUpdate() return false
      1. componentDidMount()
         1. good place to load data by instantiating the network request
         2. May call setState() immediately but it will trigger an extra rendering
    1. Updating – can be caused by changes to props or state. Methods are in the following order:
       1. GetDerivedStateFromProps()
       2. ShouldComponentUpdate(nextProps, nextState)
          1. Lets React know if a component’s output is not affected by the current changes in state or props
          2. Default behavior is re render on every state change
          3. Not called for initial render
          4. Do not rely on it to prevent a rendering
       3. Render()
       4. getSnapshotBeforeUpdate()
       5. componentDidUpdate(prevProps, prevState, snapshot)
          1. not called for initial render
          2. good place to do network requests as long as youcompare the current props to previous props
          3. may call setState immediately but it must be wrapped in a condition
    2. Unmounting – when a component is being removed from the DOM
       1. componentWillUnmount()
          1. perform any necessary clean up
          2. invalidating timers
          3. canceling network requests
          4. NOT call setState()
  1. Handling Events
     1. camelCase and with JSX pass a function as the event handler
     2. cannot return false to prevent default behavior
        1. call preventDefault
     3. Binding
        1. This.function = this.function.bind(this);
        2. In the constructor to make ‘this’ work in the callback
     4. Passing arguments to Event Handlers
        1. E is the event handler
        2. (e) => This.function(id, e) OR
        3. This.function.bind(this, id)
  2. Conditional Rendering
     1. In render() have a variable have the JSX from an if statement then return that variable in curly braces
     2. Inline if-else
        1. Condition ? true : false
     3. Prevent component form rendering
        1. Return null
  3. Lists and Keys
     1. Rendering Multiple Components
        1. Map to a JSX elements from a list and return in ol or ul
     2. Basic List Component
        1. Li you need a key
     3. Keys
        1. Help react id which items have changed, added or removed
     4. Extracting Components with Keys
        1. Elements inside the map() call need keys other wise no
     5. Keys Must Only Be Unique Among Siblings
        1. If need the same value in component pass it explicitly as a prop with a different name
     6. Embedding map() in JSX
  4. Forms
     1. Controlled Components
        1. In constructor need this.state = {value: ‘ ‘};
        2. Need a handleChange function
           1. Sets the state of the value with event.target.value
        3. Need a handleSubmit function
           1. Event.preventDefault()
        4. Form – onSubmit = this.handleSubmit
        5. Input – value = this.state.value onChange= this.handleChange
     2. The textarea tag
        1. Uses state.value and a handleChange for onChange
     3. Select tag – creates a dropdown list
        1. Select value=this.state.value onchange=this.handleChange
     4. File input tag
        1. Uncontrolled component because it is read only
     5. Handling Multiple Inputs
        1. Just also name the input and pass it to the HandleinputChange function
     6. Controlled Input Null Value
     7. Alternatives to Controlled Components
  5. Lifting State Up
     1. Rely o top-down data flow
  6. Composition vs Inheritance – they prefer composition
     1. Containment
        1. Some components don’t know their children ahead of time
     2. Specialization
        1. Have a component be a specific component
  7. Thinking In React
     1. Break the UI Into a Component Hierarchy
     2. Build a static version in REACT
        1. Don’t use state at all
        2. Pass by props
     3. ID the minimal but complete representation of UI state
     4. ID where your state should live
     5. Add inverse data flow
        1. Form components