* ReactJs makes use of JSX instead of HTML

**Components : Functions vs Classes**

React Hooks :

**Public :** folder has single page , index.html file which represent single page application.

**Src**  : react folder holds all components & state used in reactjs.

It has **index.js** which is the entry point for react.

Inside the src folder, it contains **index.js** which contains render() file which holds, ‘root’ & we are inserting App into root div.

**Note :** app is the root component.

App.js contains root component, footer header will its part or, will be inside it.

**Think every thing in terms of the component in react js.**

In reactJs we use JSX as markup than HTML.

In JSX there is some difference than HTML :

1. In HTML we write element as

<element class=”class name” >

But in JSX :

<element className = “class name”>

1. In HTML for label we write **for** as

<label for=”name of element”>

But in JSX :

<label htmlFor = “name of element”>

**App.js**

Function App(){

}

It is function, which acts as component, class can

Also act as component.

**Note :** function App() can return only one element, though inside it another element may exist.

In function app() return div / content , javascript code can be written inside the curly braces.

{ js code}

**Note** : think your app as bunch of containers.

Type **rafce** to create boiler plate for arrow function based components.

**To create class based components :**

import React from 'react'

class App extends React.Component{

render(){

return <h1>Hello from Class</h1>

}

}

export default App;

**Note : props ,** parameters passed to components.

**propTypes :** it specifies what will be the type of props, whether it will

String, integer, boolean etc . It makes the code more robust.

We can use TypeScript too, to check if type of data is correct or , not.

**Note :** while writing boolean or integer in the component make sure to write

Them within {}.

Components in ReactJS is added like XML e.g. :

< **Component />**

**Styling in React js (CSS):**

1. Use stylesheet
2. Style component
3. Direct css in javascript

**Inline styling of components :**

<h1 style={{color: 'red' , backgroundColor:'black'}}>Task Tracker {title}</h1>

**Note :** we use double bracket {{}}

We can also create an object and make **style = {headObject}**

To add onClick function on button you can use :

const Button = ({ color, text }) => {

**const onClick = () => {**

**console.log("Click");**

**};**

return (

<button

onClick={onClick}

style={{ backgroundColor: color }}

className="btn"

>

{text}

</button>

);

};

**Note :** Above code, make button less flexible by fixing the onclick function.

Other way to make onClick button more user defined / reusable is by providing onClick as parameter like a **props.**

**Task components :**

Why are we using tasks array by implementing useState ?

Since useState if defined within the Tasks.js it will have its scope

Within it only. We may need in other component, then, it is better to

Create at app level & pass the task as **props** to the **Tasks component.**

**Challenge :**  is to remove added items from the list ? Since

App.js has array of items listed, and its in the upper level of hierarchy , how to access it & updates task list.

One solution is to create a function at App.js level & pass that function as **prop** to the task.

Then, from inner level to outer level connection is made.

Array.filter() : a function is passed and that is used to judge whether to filter an element or not, if any element satisfy the condition then, it gets filtered / removed.

**Note :** that on refreshing, list restores to original state, because that is the job of backend. React is responsible for UI only. To make it persistent you may need to take help of backend.

**Learning :**

tasks.map(

(task) =>

task.id === id ? { ...task, reminder: !task.reminder } : task

)

Above syntax checks the id, if it matches then, it copies all the field of the task object & toggles ( inverts ) the value of reminder fields. It is copying object task by spreading operator.

Above code can be written as :

tasks.map(

(task) => {

if (task.id === id) {

console.log(`task ${task.id} have been toggled`);

return { ...task, reminder: !task.reminder };

} else {

return task;

}

}

// task.id === id ? { ...task, reminder: !task.reminder } : task

)

**Learning :**

**{ }** is used to denote an expression & when some evaluation is needed to be done then use following :

`${expression to evaluate}`

Each of task components are considered as **hook.**

**Challenge : How to add new dynamic tasks ? Via add button ?**

My approach would be to create a component of type **task**  and add it to the **tasks component.**

So, a new object is also needed to be added into the array ( hook ) of task.

First Create a form to take details

Second, on submit add an event listener to parse the data.

**Note :**

Inside the AddTask component , useState hook has been used to store data of the form in state variable.

However, useState variable is accessible to AddTask components only, its not available at app level.git

**Doubt : why there is a need of creating a useState variable inside the AddTask component.**

When I assigned **onSubmit** event to the form then, it became functional. Why it did’nt work when it was assigned to **input** of type=’submit’ ? Since it’s function is also to accept the value & submit , it should also work.

-> The onsubmit attribute fires when a form is submitted.

-> **<input type="submit">**

[<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) elements of type **submit** are rendered as buttons. When the [click](https://developer.mozilla.org/en-US/docs/Web/API/Element/click_event) event occurs (typically because the user clicked the button), the [user agent](https://developer.mozilla.org/en-US/docs/Glossary/User_agent) attempts to submit the form to the server.

Here **input** onsubmit sends the file to server, whereas form onsubmit executes the desired function provided.   
So what does the onSubmit of input actually does ?

**Note :** input type = ‘submit’ **it does not have any attribute onSubmit. :)z**

**Doubt :** what does **onAdd** keyword ?

~~Most probably its an event listener . But clarity is needed & doubt still exists.~~ **No, is not an event listener , rather it has been used as a props.**

**Challenge : Some problem is with associating onAdd & onSubmit functionality to buttons of the form**

**Challenge :** how to append the tasks to **tasks** array ?

// Add Task

const addTask = (task) => {

console.log(task);

console.log(tasks);

setTasks((tasks) => [...tasks, task]);

console.log(tasks);

};

The above code adds the task to array but there is some problem, like its key id needs to be added .   
There is one more peculiar problem that state of all newly added tasks are governed by each of the newly added tasks .

Like if 2 tasks are added , toggling state from reminder true to false, it changes the state of both.

**The above problem got resolved** by adding unique key to each of the components.

Syntax to append an object to task array is as follows :

**setTasks([...tasks, newTask]);**

**setTasks((tasks)=>[..tasks, newTask]);**

**Doubt: What is the difference between the two ways of doing task ?**

**Challenge :**

**Code functionality for Add button which show & close addTask functions.**

**Challenge :**

**When passing function as props to inner components why we**

**Pass as onClick={function name}** & at last level **syntax is as follows :**

**onClick={()=> {**

**function\_name(arguments);}**

**}**

=====================================================================

**Doubts in task components :**

What is key in heading tag, <h1> ?

Why it should be unique?

Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity:

=====================================================================

**Home work :**

Fect API and making HTTP request. // Travesty Media has crash course on these.

Http request : <https://www.youtube.com/watch?v=iYM2zFP3Zn0> ( done till 16:33 ) beyond this is implementation with post man and express js.

-> helps to communicate with web servers & clients

HTTP request / responses

-> http request is stateless , it does not remember past transaction, you may use cache & others to make it remember user behaviour , but at its core it is stateless.

**GET**  **:** retrieves data from the server

**POST :** submit data to the server ,

**PUT :** update data which is already on the server

**DELETE :** deletes data from the server

**HTTP header fields :**

-> it contains method, path & protocol

General header :, Response header:, Request header :

**Status code is part of header file , it is one of the most important topics.**

**Status code :**

**1xx:** Informational ( request received / processing )

**2xx: Success (** successfully received , understood and accepted )

**3xx: Redirect**

**4xx: Client error (**request does not have what it needs)

**5xx: Server error**

**Note : 1xx means from 100 to 199**

Some common status code :

**200:** Ok

**201:** Ok created

**301:** Moved to new url

**304:** Not modified ( cached version)

**400:** Bad request

**401:** Unauthorized

**404:** Not found

**500:** Internal server error

**HTTP/1.1 vs HTTP/2**

-> version 2 is more efficient & faster & has lower latency

=====================================================================

**Fetch API** :

Resource : <https://www.youtube.com/watch?v=Oive66jrwBs> , when time is available then, do

Hands on this. Practical implementation & usage has been explained.

The Fetch API provides an interface for fetching resources (including across the network).

The fetch() method takes one mandatory argument, the path to the resource you want to fetch. It returns a [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) that resolves to the [Response](https://developer.mozilla.org/en-US/docs/Web/API/Response) to that request, whether it is successful or not. You can also optionally pass in an init options object as the second argument (see [Request](https://developer.mozilla.org/en-US/docs/Web/API/Request)).

[mdn resource on it](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)

=====================================================================

**What is render() ?**

English meaning : provide or give (a service, help, etc.).

Render is a function in reactjs which helps to add element in to ReactDOM.

Everything inside the **root** div is managed by ReactDOM.

To add element in react dom, use .react() function. It takes two arguments first is **element** and second is reference to **root** .

=====================================================================

**Context API**

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

In a typical React application, data is passed top-down (parent to child) via props, but such usage can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

[documentation resouces](https://reactjs.org/docs/context.html)

Basically it makes data sharing efficient & more easier than using **props**.

=====================================================================

, **state manager Redux** .

Redux is a predictable state container designed to help you write JavaScript apps that behave consistently across client, server, and native environments and are easy to test.

With Redux, the state of your application is kept in a store, and each component can access any state that it needs from this store.

=====================================================================

**When to use {} ? What is its significance ?**

Curly braces { } are special syntax in JSX. It is used to evaluate a JavaScript expression during compilation.

USe of {} : -

* Evaluating a JavaScript variable
* Evaluating a JavaScript variable

Use of () :

* Parenthesis are used in an arrow function to return an object.

Parenthesis are used to group multiline of codes on JavaScript return statement so to prevent semicolon inserted automatically in the wrong place.

() => ({ name: 'Amanda' }) // Shorthand to return an object

() => ({ name: 'Amanda' }) // Shorthand to return an object

[**Good blog which explains well about {} and () in react**](https://javascript.plainenglish.io/curly-braces-versus-parenthesis-in-reactjs-4d3ffd33128f)

[**When to use and not use {} in react**](https://dev.to/jasterix/when-do-you-actual-need-to-use-curly-braces-in-react-25mh)

=====================================================================

**What is useState ?**

To understand useState() first, note that class based components are difficult manage,but function component does not allow to have variables whose value persists.

To solve this problem react introduced Hooks.

**Its let you use state and side effects in a functional components**, something what was not possible before.

useState() is a function which helps to create state variable & function to update those variable .

**Syntax**

const [count, setCount] = useState();

**How to initialize count variable ?**

Either you can pass hardcoded value as argument to useState() function or, as props

const [count, setCount] = useState(props.initialValue);

const [count, setCount] = useState(0);

### **How do I handle multiple state variables?**

This is another cool thing about hooks. We can have as many as we like in a component:

[hooks in react](https://www.freecodecamp.org/news/beginners-guide-to-using-react-hooks/)

[hooks in react 2](https://www.freecodecamp.org/news/react-hooks-in-5-minutes/)

**Learning :**

Syntax to copy object via destructuring is as follow :

Const **newObj = { newField, ...oldObj}**

**Next learning :**

Learn : useEffect : it is one of the hooks of react

**Shortcut :**

To increase font size of google docs : ctrl + shift + >

To decrease : ctrl + shift + <

To load template for components type hot key : **rafce** .

**Resume from : 57:00**