**# Exam**

**## About:**

- Date: Tuesday (25/10/2022)

- Time: 10:00 - 13:00

- Duration: 150 min

- Place: online, same Zoom link

> If you are not comfortable using the webcam, you have the possibility to take the exam in Espoo's Campus.

> Please ensure that you have the basic tools in place:

- A Web camera

- Microphone

**## Practice**

**### Exercise 1**

Simplify the component by using destructuring:

1.

```js

const Display = (props) => {

  return (

    <div>{props.counter}</div>

  )

}

```

2.

```js

const Button = (props) => {

  return (

    <button onClick={props.onClick}>

      {props.text}

    </button>

  )

}

```

**### Exercise 2**

1.

In the following application, define the new state object by using the object spread  syntax:

```js

export default function App() {

  const [clicks, setClicks] = useState({

    left: 0, right: 0

  })

  const handleLeftClick = () => {

    const newClicks = {

      left: clicks.left + 1,

      right: clicks.right

    }

    setClicks(newClicks)

  }

  const handleRightClick = () => {

    const newClicks = {

      left: clicks.left,

      right: clicks.right + 1

    }

    setClicks(newClicks)

  }

  return (

    <div>

      {clicks.left}

      <button onClick={handleLeftClick}>left</button>

      <button onClick={handleRightClick}>right</button>

      {clicks.right}

    </div>

  )

}

```

2. In the code above, why didn't we update the state directly, as follows:

```js

const handleLeftClick = () => {

  clicks.left++

  setClicks(clicks)

}

```

**### Exercise 3**

Given the following snippet

```js

let fruitStateVariable = useState('banana');

```

1. What did we pass to useState as an argument?

2. What type is `fruitStateVariable`

3. What type is `fruitStateVariable[1]`

**### Exercise 4**

What is wrong with the following React component? How can you fix it?

```js

export default function App() {

  const [age, setAge] = useState(0)

  if ( age > 10 ) {

    const [foobar, setFoobar] = useState(null)

  }

  return (

    //...

  )

}

```

**### Exercise 5**

Explain the definition of the `notesToShow` variable.

```js

const notesToShow = showAll

  ? notes

  : notes.filter(note => note.important === true)

```

**### Exercise 6**

What is wrong with the following code?

```js

function welcome(props) {

  return <h1>Hello, {props.name}</h1>;

}

export default function App() {

  return (

    <div>

      <welcome name="Sara" />

      <welcome name="Cahal" />

    </div>

  );

}

```

**### Exercise 7**

The following figure is for a Counter.

![](/counter.png)

- When you click the `+` button, the counter uses the value in the input text field to increment `counterValue`.

- When you click the `-` button, the counter uses the value in the input text field to decrement `counterValue`.

- When `counterValue` is larger than 100, it will be displayed in green.

- When `counterValue` is less than -100, it will be displayed in red.

Use the code below as a starting  point and fill in the **\*\*4 place holders\*\***.

```js

import { useState } from "react";

function Counter() {

  const [counterValue, setCounterValue] = useState(0);

  const [inputValue, setInputValue] = useState(1);

  const addToCounter = () => {

    /\* Place holder 2 \*/

  }

  const subFromCounter = () => {

    /\* Place holder 3 \*/

  }

  return (

    <div className="text-center">

      <h3>My Counter</h3>

      <h2   className={ /\* Place holder 4 \*/ } >{counterValue}</h2>

      <button

      onClick={subFromCounter}

      >-</button>

      <input

      type="input"

      value={inputValue}

      className="text-center"

      onChange={(e)=>{

        /\* Place holder 1 \*/

      }}

      />

      <button

      onClick={addToCounter}

      >+</button>

    </div>

  );

}

export default Counter;

```

**### Exercise 8**

Explain how the following component works:

```js

const History = (props) => {

  if (props.allClicks.length === 0) {

    return (

      <div>

        the app is used by pressing the buttons

      </div>

    )

  }

  return (

    <div>

      button press history: {props.allClicks.join(' ')}

    </div>

  )

}

```

**### Exercise 9**

What's wrong with the following code?

```js

const Button = (props) => (

  <button onClick={props.handleClick}>

    {props.text}

  </button>

)

export default function App() {

  const [value, setValue] = useState(10)

  const setToValue = newValue => {

    console.log('value now', newValue)

    setValue(newValue)

  }

  const Display = props => <div>{props.value}</div>

  return (

    <div>

      <Display value={value} />

      <Button handleClick={() => setToValue(1000)} text="thousand" />

      <Button handleClick={() => setToValue(0)} text="reset" />

      <Button handleClick={() => setToValue(value + 1)} text="increment" />

    </div>

  )

}

```

**### Exercise 10**

Given the animals array below:

```js

let animals = ['cat', 'chicken', 'cow', 'sheep', 'horse']

```

Use the map function to render the elements as ollows:

```html

<ul>

<li>cat</li>

<li>chicken</li>

<li>cow</li>

<li>sheep</li>

<li>horse</li>

</ul>

```

**### Exercise 11**

Add the necessary code to render the list of users.

```js

const users = [

  { name: "John Doe", id: 1 },

  { name: "Jane Doe", id: 2 },

  { name: "Billy Doe", id: 3 }

];

const userItems = users.map(

    /\* Your code here\*/

);

export default function App() {

  return (

    <>

      <h3>Users:</h3>

      <ul>{userItems}</ul>

    </>

  );

}

```

**### Exercise 12**

1. Explain the meaning of the following sentence:

> "All React components must act like pure functions with respect to their props".

2. React was born out of Facebook solving its own problems and has since become a  standard in frontend development. What contributed to this success?

3. State the Key differences between functional programming and object oriented programming!

4.  How does React help teams, projects, and companies be more efficient?

5. When Should I Not Use React?

6. How do React elements handles Event?  Compare with DOM elements.

7. Discuss the similarities and differences between JSX and HTML.

**### Exercise 13 (ES6+)**

1. What is the difference between  Spread and Rest parameters?

2. What are the template literals?

3. Explain ES6 promises.

4. What is callback and callback hell in JavaScript?

5. What are the benefits of using spread syntax

**### Exercise 14**

Explain how the following code works!

```js

import { useState, useEffect } from 'react'

import axios from 'axios'

import Note from './components/Note'

export default function App() {

  const [notes, setNotes] = useState([])

  const [newNote, setNewNote] = useState('')

  const [showAll, setShowAll] = useState(true)

  useEffect(() => {

    console.log('effect')

    axios

      .get('http://localhost:3001/notes')

      .then(response => {

        console.log('promise fulfilled')

        setNotes(response.data)

      })

  }, [])

  console.log('render', notes.length, 'notes')

  return (

    //...

  )

}

```

**### Exercise 15**

Explain how the following code works!

```js

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

  const [post, setPost] = React.useState(null);

  React.useEffect(() => {

    axios.get(`${baseURL}/1`).then((response) => {

      setPost(response.data);

    });

  }, []);

  function createPost() {

    axios

      .post(baseURL, {

        title: "Hello World!",

        body: "This is a new post."

      })

      .then((response) => {

        setPost(response.data);

      });

  }

  if (!post) return "No post!"

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

      <button onClick={createPost}>Create Post</button>

    </div>

  );

}

```

**### Exercise 16**

Explain how the following code works!

```js

export default function App() {

  const [value, setValue] = React.useState("");

  return (

    <>

      <h3>Disable Button Challenge</h3>

      <input type="text" onChange={(e) => setValue(e.target.value)} />

      <button disabled={value.length < 1}>Submit</button>

    </>

  );

}

```

**### Exercise 17**

- [Bootcamp](<https://github.com/ironhack-labs/lab-react-training>

**### Other**

- In Class activities

- [fullstackopen.com](fullstackopen.com/)

<https://fullstackopen.com/>

**### Other Videos**

- [ES6+](<https://www.youtube.com/playlist?list=PLnHJACx3NwAfRUcuKaYhZ6T5NRIpzgNGJ>)

- [Net Ninja: Modern React Tutorial](<https://www.youtube.com/playlist?list=PL4cUxeGkcC9gZD-Tvwfod2gaISzfRiP9d>)

#1 -  Tutorial#13

Answers:

**## Practice**

**### Exercise 1**

Simplify the component by using destructuring:

1.

```js

const Display = (props) => {

  return (

    <div>{props.counter}</div>

  )

}

Answer:

const Display = ({ counter }) => <div>{counter}</div>

2.

```js

const Button = (props) => {

  return (

    <button onClick={props.onClick}>

      {props.text}

    </button>

  )

}

Answer:

const Button = ({ onClick, text }) => <button onClick={onClick}>{text}</button>

**### Exercise 2**

1.

In the following application, define the new state object by using the object spread  syntax:

```js

export default function App() {

  const [clicks, setClicks] = useState({

    left: 0, right: 0

  })

  const handleLeftClick = () => {

    const newClicks = {

      left: clicks.left + 1,

      right: clicks.right

    }

    setClicks(newClicks)

  }

  const handleRightClick = () => {

    const newClicks = {

      left: clicks.left,

      right: clicks.right + 1

    }

    setClicks(newClicks)

  }

  return (

    <div>

      {clicks.left}

      <button onClick={handleLeftClick}>left</button>

      <button onClick={handleRightClick}>right</button>

      {clicks.right}

    </div>

  )

}

Answer1

const handleLeftClick = () => {

    const newClicks = {

      ...clicks,

      left: clicks.left + 1

    }

    setClicks(newClicks)

  }

  const handleRightClick = () => {

    const newClicks = {

      ...clicks,

      right: clicks.right + 1

    }

    setClicks(newClicks)

  }

Answer2 more short:

const handleLeftClick = () =>

setClicks({ ...clicks, left: clicks.left + 1 })

const handleRightClick = () =>

  setClicks({ ...clicks, right: clicks.right + 1 })

2. In the code above, why didn't we update the state directly, as follows:

```js

const handleLeftClick = () => {

  clicks.left++

  setClicks(clicks)

}

```

Answer:

The application appears to work. However, *it is forbidden in React to mutate state directly*, since [it can result in unexpected side effects](https://stackoverflow.com/a/40309023). Changing state has to always be done by setting the state to a new object. If properties from the previous state object are not changed, they need to simply be copied, which is done by copying those properties into a new object, and setting that as the new state.

Storing all of the state in a single state object is a bad choice for this particular application; there's no apparent benefit and the resulting application is a lot more complex. In this case storing the click counters into separate pieces of state is a far more suitable choice.

There are situations where it can be beneficial to store a piece of application state in a more complex data structure. [The official React documentation](https://reactjs.org/docs/hooks-faq.html#should-i-use-one-or-many-state-variables) contains some helpful guidance on the topic.

**### Exercise 3**

Given the following snippet

```js

let fruitStateVariable = useState('banana');

```

1. What did we pass to useState as an argument?

2. What type is `fruitStateVariable`

3. What type is `fruitStateVariable[1]`

import React, { useState } from 'react'

const App = () => {

let fruitStateVariable = useState('banana');

console.log(typeof fruitStateVariable)

console.log(typeof fruitStateVariable[1])

}

export default App

1 ‘banana’

2, object

3. function

**### Exercise 4**

What is wrong with the following React component? How can you fix it?

```js

export default function App() {

  const [age, setAge] = useState(0)

  if ( age > 10 ) {

    const [foobar, setFoobar] = useState(null)

  }

  return (

    //...

  )

}

```

ANSWER:

**Rules of Hooks**

There are a few limitations and rules we have to follow to ensure that our application uses hooks-based state functions correctly.

The useState function (as well as the useEffect function introduced later on in the course) *must not be called* from inside of a loop, a conditional expression, or any place that is not a function defining a component. This must be done to ensure that the hooks are always called in the same order, and if this isn't the case the application will behave erratically.

To recap, hooks may only be called from the inside of a function body that defines a React component:

const App = () => {

// these are ok

const [age, setAge] = useState(0)

const [name, setName] = useState('Juha Tauriainen')

if ( age > 10 ) {

// this does not work!

const [foobar, setFoobar] = useState(null)

}

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

// also this is not good

const [rightWay, setRightWay] = useState(false)

}

const notGood = () => {

// and this is also illegal

const [x, setX] = useState(-1000)

}

return (

//...

)

}

**### Exercise 5**

Explain the definition of the `notesToShow` variable.

```js

const notesToShow = showAll

  ? notes

  : notes.filter(note => note.important === true)

```

Answer:

    const notesToShow = showAll

        ? notes

        : notes.filter(note => note.important === true)

The operator functions as follows. If we have:

const result = condition ? val1 : val2

the result variable will be set to the value of val1 if condition is true. If condition is false, the result variable will be set to the value ofval2.

**### Exercise 6**

What is wrong with the following code?

```js

function welcome(props) {

  return <h1>Hello, {props.name}</h1>;

}

export default function App() {

  return (

    <div>

      <welcome name="Sara" />

      <welcome name="Cahal" />

    </div>

  );

}

```

Function must start with uppercase lettet

This works:

function Welcome(props) {

  return <h1>Hello, {props.name}</h1>;

}

export default function App() {

  return (

    <div>

      <Welcome name="Sara" />

      <Welcome name="Cahal" />

    </div>

  );

}

**### Exercise 7**

The following figure is for a Counter.

![](/counter.png)

- When you click the `+` button, the counter uses the value in the input text field to increment `counterValue`.

- When you click the `-` button, the counter uses the value in the input text field to decrement `counterValue`.

- When `counterValue` is larger than 100, it will be displayed in green.

- When `counterValue` is less than -100, it will be displayed in red.

Use the code below as a starting  point and fill in the **\*\*4 place holders\*\***.

```js

import { useState } from "react";

function Counter() {

  const [counterValue, setCounterValue] = useState(0);

  const [inputValue, setInputValue] = useState(1);

  const addToCounter = () => {

    /\* Place holder 2 \*/

  }

  const subFromCounter = () => {

    /\* Place holder 3 \*/

  }

  return (

    <div className="text-center">

      <h3>My Counter</h3>

      <h2   className={ /\* Place holder 4 \*/ } >{counterValue}</h2>

      <button

      onClick={subFromCounter}

      >-</button>

      <input

      type="input"

      value={inputValue}

      className="text-center"

      onChange={(e)=>{

        /\* Place holder 1 \*/

      }}

      />

      <button

      onClick={addToCounter}

      >+</button>

    </div>

  );

}

export default Counter;

```

Failed:

import { useState } from "react";

function App() {

  const [counterValue, setCounterValue] = useState(0);

  const [inputValue, setInputValue] = useState(1);

  const addToCounter = () => {

    /\* Place holder 2 \*/

    setCounterValue(counterValue  + 1)

  }

  const subFromCounter = () => {

    /\* Place holder 3 \*/

    setCounterValue(counterValue - 1)

  }

  return (

    <div className="text-center">

      <h3>My Counter</h3>

      <h2>{counterValue}</h2>

      <button

      onClick={subFromCounter}

      >-</button>

      <input

      type="input"

      value={inputValue}

      className="text-center"

      onChange={(e)=>{

        /\* Place holder 1 \*/

        setInputValue(e.target.value)

      }}

      />

      <button

      onClick={addToCounter}

      >+</button>

    </div>

  );

}

export default App;

**### Exercise 8**

Explain how the following component works:

```js

const History = (props) => {

  if (props.allClicks.length === 0) {

    return (

      <div>

        the app is used by pressing the buttons

      </div>

    )

  }

  return (

    <div>

      button press history: {props.allClicks.join(' ')}

    </div>

  )

}

```

See

React\_P1\_4\_More\_complex\_state\_debugging

**### Exercise 9**

What's wrong with the following code?

```js

const Button = (props) => (

  <button onClick={props.handleClick}>

    {props.text}

  </button>

)

export default function App() {

  const [value, setValue] = useState(10)

  const setToValue = newValue => {

    console.log('value now', newValue)

    setValue(newValue)

  }

  const Display = props => <div>{props.value}</div>

  return (

    <div>

      <Display value={value} />

      <Button handleClick={() => setToValue(1000)} text="thousand" />

      <Button handleClick={() => setToValue(0)} text="reset" />

      <Button handleClick={() => setToValue(value + 1)} text="increment" />

    </div>

  )

}

What is wrong`?

import { useState } from 'react'

is missing from start

AND:

import { useState } from 'react'

// This is the right place to define a component

const Button = (props) => (

  <button onClick={props.handleClick}>

    {props.text}

  </button>

)

const App = () => {

  const [value, setValue] = useState(10)

  const setToValue = newValue => {

    console.log('value now', newValue)

    setValue(newValue)

  }

  // Do not define components inside another component

  const Display = props => <div>{props.value}</div>

  return (

    <div>

      <Display value={value} />

      <Button handleClick={() => setToValue(1000)} text="thousand" />

      <Button handleClick={() => setToValue(0)} text="reset" />

      <Button handleClick={() => setToValue(value + 1)} text="increment" />

    </div>

  )

}

export default App

The application still appears to work, but **don't implement components like this!** Never define components inside of other components. The method provides no benefits and leads to many unpleasant problems. The biggest problems are due to the fact that React treats a component defined inside of another component as a new component in every render. This makes it impossible for React to optimize the component.

Let's instead move the *Display* component function to its correct place, which is outside of the *App* component function:

const Display = props => <div>{props.value}</div>

const Button = (props) => (

<button onClick={props.handleClick}>

{props.text}

</button>

)

const App = () => {

const [value, setValue] = useState(10)

const setToValue = newValue => {

console.log('value now', newValue)

setValue(newValue)

}

return (

<div>

<Display value={value} />

<Button handleClick={() => setToValue(1000)} text="thousand" />

<Button handleClick={() => setToValue(0)} text="reset" />

<Button handleClick={() => setToValue(value + 1)} text="increment" />

</div>

)

}

**### Exercise 10**

Given the animals array below:

```js

let animals = ['cat', 'chicken', 'cow', 'sheep', 'horse']

```

Use the map function to render the elements as follows:

```html

<ul>

<li>cat</li>

<li>chicken</li>

<li>cow</li>

<li>sheep</li>

<li>horse</li>

</ul>

```

ANSWER:

// added key

export default function App() {

  let animals = ['cat', 'chicken', 'cow', 'sheep', 'horse']

return (

  <div>

   <ul>{animals.map(note => <li key = {note} >{note}</li>)}

  </ul>

  </div>

)

}

**### Exercise 11**

Add the necessary code to render the list of users.

```js

const users = [

  { name: "John Doe", id: 1 },

  { name: "Jane Doe", id: 2 },

  { name: "Billy Doe", id: 3 }

];

const userItems = users.map(

    /\* Your code here\*/

);

export default function App() {

  return (

    <>

      <h3>Users:</h3>

      <ul>{userItems}</ul>

    </>

  );

}

Answer:

// for exam study exercise 11

const users = [

  { name: "John Doe", id: 1 },

  { name: "Jane Doe", id: 2 },

  { name: "Billy Doe", id: 3 }

];

const userItems = users.map(item =>

  <li key= {item.id}>{item.name}</li>)

    /\* Your code here\*/

export default function App() {

  return (

    <>

      <h3>Users:</h3>

      <ul>{userItems}</ul>

    </>

  );

}

**### Exercise 12**

1. Explain the meaning of the following sentence:

> "All React components must act like pure functions with respect to their props".

2. React was born out of Facebook solving its own problems and has since become a  standard in frontend development. What contributed to this success?

3. State the Key differences between functional programming and object oriented programming!

4.  How does React help teams, projects, and companies be more efficient?

5. When Should I Not Use React?

6. How do React elements handles Event?  Compare with DOM elements.

7. Discuss the similarities and differences between JSX and HTML.

1. Explain the meaning of the following sentence:

Why all React components must act like pure functions with respect to their props?

Props are Read-Only  
  
Such functions are called “pure” **because they do not attempt to change their inputs, and always return the same result for the same inputs**. React is pretty flexible but it has a single strict rule: All React components must act like pure functions with respect to their props

2. React was born out of Facebook solving its own problems and has since become a  standard in frontend development. What contributed to this success?

**Facebook Ads became hard to manage, so Facebook needed to come up with a good solution for it**. Jordan Walke worked on the prototype and created React.31.5

3. State the Key differences between functional programming and object oriented programming!

| Functional Programming | Object Oriented Programming |
| --- | --- |
| This programming paradigm emphasizes on the use of functions where each function performs a specific task. | This programming paradigm is based on object oriented concept. Classes are used where instance of objects are created |
| Fundamental elements used are variables and functions.The data in the functions are immutable(cannot be changed after creation). | Fundamental elements used are objects and methods and the data used here are mutable data. |
| Importance is not given to data but to functions. | Importance is given to data rather than procedures. |
| It follows declarative programming model. | It follows imperative programming model. |
| It uses recursion for iteration. | It uses loops for iteration. |
| It is parallel programming supported. | It does not support parallel programming. |
| The statements in this programming paradigm does not need to follow a particular order while execution. | The statements in this programming paradigm need to follow an order i.e., bottom up approach while execution. |
| Does not have any access specifier. | Has three access specifiers namely, Public, Private and Protected. |
| To add new data and functions is not so easy. | Provides an easy way to add new data and functions. |
| No data hiding is possible. Hence, Security is not possible. | Provides data hiding. Hence, secured programs are possible. |

4.  How does React help teams, projects, and companies be more efficient?

How React is efficient?

Internally, **React uses several clever techniques to minimize the number of costly DOM operations required to update the UI**. For many applications, using React will lead to a fast user interface without doing much work to specifically optimize for performance.

Because it uses the declarative approach, it's generally **easier to follow and understand the code**. Thus, it makes it easier to integrate people into the team and collaborate on the application project. One of the many benefits of React.

Because **ReactJs is one of the most widely used front-end JavaScript libraries for creating Web Applications**. Around 8,000 businesses around the world have chosen React over other popular libraries and frameworks for building rich user interfaces.

React is **a JavaScript library developed by Facebook which, among other things, was used to build Instagram.com**. Its aim is to allow developers to easily create fast user interfaces for websites and applications alike. The main concept of React. js is virtual DOM.15.3.2022

5. When Should I Not Use React?

1. **Complex Interfaces**. If you are looking for building a mobile app with multiple screen transitions, interactions and animations, React Native is probably not the best bet for your mobile app development project. Generally, mobile apps with need of complex gestures are not built with React Native.18.

Why founders should NOT use React Native:

1. Animation is \*always\* choppy due to the screen refreshing

2. The best & most affordable developers use Swift & Kotlin

3. The support for Android is terrible

4. Huge security issues with the JS pipeline Write once, ruin everywhere

6. How do React elements handles Event?  Compare with DOM elements.

Is event handling in React is similar to event handling in DOM?

**Handling events with React elements is very similar to handling events on DOM elements**. There are some syntax differences: React events are named using camelCase, rather than lowercase. With JSX you pass a function as the event handler, rather than a string.

In React, all DOM properties and attributes (including event handlers) should be camelCased. For example, the HTML attribute tabindex corresponds to the attribute tabIndex in React. The exception is aria-\* and data-\* attributes, which should be lowercased. For example, you can keep aria-label as aria-label.

7. Discuss the similarities and differences between JSX and HTML.

The basic differences between JSX and HTML are as follows:

|  |  |
| --- | --- |
| HTML | JSX |
| In HTML, multiple elements can be returned. For example: *<ul>* *<li>unordered list* *<ol>* *<li>ordered list</li>* *<li>ordered list</li>* *<li>ordered list</li>* *</ol>* *</li>* *<li>unordered list</li>* *<li>unordered list</li>* *</ul>* | Nested JSX must return one element, which we’ll call a parent element that wraps all other levels of nested elements: *<div>* *<p>pink</p>* *<p>yellow</p>* *<p>green</p>* *</div>* Without the wrapper element, JSX won’t transpile. In React, we can render JSX directly into HTML DOM using React rendering API, aka *ReactDOM*. The formula for rendering React elements seems like this: *ReactDOM.render(componentToRender, targetNode)* *ReactDOM.render()* must be called after the JSX elements declarations. |
| HTML elements have attributes.  e.g., *maxlength* in *<input maxlength=”16″ />* | JSX elements have props.  e.g., *maxLength* in *<input maxLength=”16″ />* |
| It is not necessary to use camelCase for attributes, ids and event references. Its totally your call to use camelCase, lowercase or hyphens for naming them. | All HTML attributes and event references in JSX become camelCase, this way, *onclick* event becomes*onClick* and *onchange* — *onChange.* |
| The class attribute can be used on any HTML element. The class name can be used by CSS and JavaScript to perform certain tasks for elements with the specified class name. | You can’t use the word *class* to define HTML classes, since *class* is a reserved word in JavaScript, instead, use — *className*. |
| In HTML almost all tags have an opening and a closing tag except probably a few like  *<br/>* | In JSX, however, any element can be written as a self-closing tag, for example:*<div/>* Example: *const string = <img src={user.avatarUrl}  />;* |

Since the JSX component represents HTML, you can put several components together to make a more complex HTML page.

The fact that JSX looks like HTML doesn’t make it any more of HTML, in fact, you can still write normal functions bypassing the HTML-like syntax.

**### Exercise 13 (ES6+)**

1. What is the difference between  Spread and Rest parameters?

2. What are the template literals?

3. Explain ES6 promises.

4. What is callback and callback hell in JavaScript?

5. What are the benefits of using spread syntax

1. What is the difference between  Spread and Rest parameters?

The spread operator allows us to spread the value of an array (or any iterable) across zero or more arguments in a function or elements in an array (or any iterable). The rest parameter allows us to pass an indefinite number of parameters to a function and access them in an array.

2. What are the template literals?

Template literals are **literals delimited with backtick ( ` ) characters, allowing for multi-line strings, string interpolation with embedded expressions, and special constructs called tagged templates**.

You can use ${ }

**Template literals** provide an easy way to interpolate variables and expressions into strings.

The method is called string interpolation.

The syntax is:

${...}

3. Explain ES6 promises.

Promises are a way to implement asynchronous programming in JavaScript(ES6 which is also known as ECMAScript-6). A Promise acts as a container for future values. Like if you order any food from any site to deliver it to your place that order record will be the promise and the food will be the value of that promise. So the order details are the container of the food you ordered. Let’s explain it with another example. You order an awesome camera online. After your order is placed you receive a receipt of the order. That receipt is a Promise that your order will be delivered to you. The receipt is a placeholder for the future value namely the camera. Promises used in JavaScript for asynchronous programming. For asynchronous programming, JavaScript uses [callbacks](https://www.geeksforgeeks.org/javascript-callbacks/), but there is a problem using the callback which is callback hell (multiple or dependent callbacks) or Pyramid of Doom. Using the ES6 Promise will simply avoid all the problems associated with the callback.

4. What is callback and callback hell in JavaScript?

[Callback](https://www.geeksforgeeks.org/javascript-callbacks/)**:**A callback is a function that is passed as an argument to another function that executes the callback based on the result. They are basically functions that are executed only after a result is produced. Callbacks are an important part of asynchronous JavaScript.

**Example:**

* Javascript

|  |
| --- |
| // Main function  const mainFunction = (callback) => {      setTimeout(() => {          callback([2, 3, 4]);      }, 2000)  }    // Add function  const add = (array) => {      let sum = 0;  **for**(let i of array) {          sum += i;      }      console.log(sum);  }    // Calling main function  mainFunction(add); |

**Output:**

9

**Callback Hell:**Callback Hell is essentially nested callbacks stacked below one another forming a pyramid structure. Every callback depends/waits for the previous callback, thereby making a pyramid structure that affects the readability and maintainability of the code.

<**script**>

    let words = document.querySelectorAll(".word");

    const animateAll = (animate) => {

        setTimeout(() => {

            animate(words[0]);

            setTimeout(() => {

                animate(words[1]);

                setTimeout(() => {

                    animate(words[2]);

                }, 1000)

            }, 1000)

        }, 1000)

    }

    const animate = (word) => {

        word.classList.add("animate");

    }

    animateAll(animate);

</**script**>

5. What are the benefits of using spread syntax

**Spread Operator:**Spread operator or **Spread Syntax**allow us to expand the arrays and objects into elements in the case of an array and key-value pairs in the case of an object. The spread syntax is represented by three dots (…) in JavaScript.

**Syntax:**

var my\_var = [...array];

**Benefits of using Spread syntax:**

1. It allows us to include all elements of an array or object in a list of some kind. It can expand an object or an array and store all values in a new variable of the same datatype

<**script**>

    // Operating spread syntax to expand an Object

    var obj1 = {

        organisation: "GFG",

        fullForm: "Geeks for Geeks"

    };

    var obj2 = {

        description: "A learning platform for Geeks."

    };

    var obj3 = {...obj2, ...obj1

    };

    console.log(obj3);

    // Operating spread syntax to expand an Array

    var fruits = ["Mango", "Banana", "Apple"];

    var moreFruits = [...arr1, "Orange",

                "Pineapple", "Watermelon"];

    console.log(moreFruits);

</**script**>

2. It allows us to copy all elements of an already existing array into a new array and perform push, pop operations without even disturbing the previous array.

**Example:**

* JavaScript

|  |
| --- |
| <**script**>      // Copying an array into another      // array and operate pop      // operation without disturbing      // data of first array      var arr1 = ['a', 'b', 'c', 'd', 'e'];      var arr2 = [...arr1];      console.log("arr1 before applying pop operation:");      console.log(arr1);      arr2.pop();      console.log("arr1 & arr2 after applying"+                  " pop operation on arr2:");      console.log(arr2);      console.log(arr1);  </**script**> |

3. It can concat a string or merge two or more arrays without using **concat()** function.

<**script**>

    // concatenation of two arrays without

    // concat function using spread syntax

    var breakfast = ["Bread", "Sandwich", "Fruits"];

    var moreBreakfast = ["Salad", "Tea & Coffee"];

    console.log("Breakfast before concatenation:");

    console.log(breakfast);

    breakfast = [...breakfast, ...moreBreakfast];

    console.log("Breakfast after concatenation:");

    console.log(breakfast);

</**script**>

4. It allows us to operate Math functions on an array. Because we can’t use Math functions directly on arrays. The spread operator expands an iterable object array into a list of arguments that allow us to operate the Math functions and get the desired result.

**Example:**

* JavaScript

|  |
| --- |
| <**script**>      // Operating an array in an Math      // function using spread operator      var arr1 = [99, 50, 130, 1, 98, 23, 66];      console.log("Maximum element of arr1 is:")      console.log(Math.max(...arr1));  </**script**> |

5. It allows us to do changes in nested object without reflecting in original object

Example :

**Structure of pointing object in nested object**

let obj = {

   name:'Geeks for Geeks',

   add:{

      country:"INDIA",

      state:{

          code:"DL",

          pincode:"129089"

      }

  }

}

let obj2 = {...obj} ///SHALLOW COPY

obj2.name = "GFG",

console.log(obj2);

obj2 = {...obj2,add:{...obj.add}}

obj2.add.country = "BHARAT"

console.log(obj2)

obj2 = {...obj2,add:{...obj2.add,state:{...obj.add.state}}}  //DEEP COPY

obj2.add.state.pincode  = 823687

console.log("original object")

console.log(obj);

console.log("doing all changes final object")

console.log(obj2);

**### Exercise 14**

Explain how the following code works!

```js

import { useState, useEffect } from 'react'

import axios from 'axios'

import Note from './components/Note'

export default function App() {

  const [notes, setNotes] = useState([])

  const [newNote, setNewNote] = useState('')

  const [showAll, setShowAll] = useState(true)

  useEffect(() => {

    console.log('effect')

    axios

      .get('http://localhost:3001/notes')

      .then(response => {

        console.log('promise fulfilled')

        setNotes(response.data)

      })

  }, [])

  console.log('render', notes.length, 'notes')

  return (

    //...

  )

}

React\_P2\_c\_getting\_data\_from\_server

**### Exercise 15**

Explain how the following code works!

```js

import axios from "axios";

import React from "react";

const baseURL = "https://jsonplaceholder.typicode.com/posts";

export default function App() {

  const [post, setPost] = React.useState(null);

  React.useEffect(() => {

    axios.get(`${baseURL}/1`).then((response) => {

      setPost(response.data);

    });

  }, []);

  function createPost() {

    axios

      .post(baseURL, {

        title: "Hello World!",

        body: "This is a new post."

      })

      .then((response) => {

        setPost(response.data);

      });

  }

  if (!post) return "No post!"

  return (

    <div>

      <h1>{post.title}</h1>

      <p>{post.body}</p>

      <button onClick={createPost}>Create Post</button>

    </div>

  );

}

At the beginning:

**sunt aut facere repellat provident occaecati excepturi optio reprehenderit**

quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto

Create Post

When you press the button:

# Hello World!

This is a new post.

Create Post

**### Exercise 16**

Explain how the following code works!

```js

export default function App() {

  const [value, setValue] = React.useState("");

  return (

    <>

      <h3>Disable Button Challenge</h3>

      <input type="text" onChange={(e) => setValue(e.target.value)} />

      <button disabled={value.length < 1}>Submit</button>

    </>

  );

}

## **Definition and Usage**

The disabled attribute is a boolean attribute.

When present, it specifies that the button should be disabled.

A disabled button is unusable and un-clickable.

The disabled attribute can be set to keep a user from clicking on the button until some other condition has been met (like selecting a checkbox, etc.). Then, a JavaScript could remove the disabled value, and make the button clickable again.

Graphical user interface, text, application, email

Description automatically generated

**### Exercise 17**

- [Bootcamp](<https://github.com/ironhack-labs/lab-react-training>