


React Practice

App Component

```
export default function App() {
  async function getAdvice() {
    const response = await fetch("https://api.adviceslip.com/advice");
    const data = await response.json();
    console.log(data);
  }

  return (
    <div>
      <h1>Hello World!</h1>
      <button onClick={getAdvice}>Get Advice!</button>
    </div>
  );
}
```



```
import { useEffect, useState } from "react";
```

```
export default function App() {
  /* useState is a function that returns an array in react
  and here we are destructuring that array 1st position of the
  array is value of the state, the 2nd value is a setter function that
  we can use to update the piece of state */
  const [advice, setAdvice] = useState("");
  const [count, setCount] = useState(0);

  async function getAdvice() {
    const res = await fetch("https://api.adviceslip.com/advice");
    const data = await res.json();
    setAdvice(data.slip.advice);
    /* Now in this getAdvice function we can use setAdvice function to
    update the state, whenever the piece of state is updated
    user interface will also be updated */
    // take current count add 1 and that will become the new count
    setCount((c) => c + 1);
  }

  // Generate very first piece of advice when loaded
  /* useEffect takes two arguments 1st a function that
  we want to execute at the beginning when the component loads and 2nd
  argument is dependency array */
}
```

```

useEffect(function () {
  getAdvice();
}, []);
return (
  <div>
    <h1>{advice}</h1>
    <button onClick={getAdvice}>Get Advice!</button>
    {/* include this component like it is another html element */}
    {/* Pass count as a prop to Message. Props are like parameters
    to function. We call the prop count and pass in the prop
    value */}
    <Message count={count} />
  </div>
);
}

// In react we divide user interfaces into components
// components are reusable pieces of code
// components are used to render UI
// Name of all components should start with capital letter(convention)
// Now accept the props object as a parameter. In this props object count is
now a property
function Message(props) {
  // update count dynamically
  return (
    <p>
      You have read <strong>{props.count}</strong> pieces of advice
    </p>
  );
}

```

JSX Template: Babel transpiles JSX code by transforming it into regular JavaScript code. JSX (JavaScript XML) is a syntax extension for JavaScript that allows developers to write HTML-like code within JavaScript. Babel is a toolchain that is primarily used to convert ECMAScript 2015+ (ES6+) code into a backwards-compatible version of JavaScript that can be run in older browsers or environments.

When Babel encounters JSX code, it transforms it into function calls that create React elements. Here's a simplified explanation of how Babel transpiles JSX:

1. ****Parsing****: Babel first parses the JSX syntax into an Abstract Syntax Tree (AST), which is a structured representation of the code.

2. **Transformation**: Babel then transforms the JSX AST into equivalent JavaScript code. This transformation involves replacing JSX elements with `React.createElement()` function calls.

3. **Generation**: Finally, Babel generates the transpiled JavaScript code from the transformed AST.

For example, given the following JSX code:

```
```jsx
const element = <h1>Hello, world!</h1>;
```
```

Babel transpiles it into the following JavaScript code:

```
```javascript
const element = React.createElement("h1", null, "Hello, world!");
```
```

In this transpiled code, `React.createElement()` is used to create a React element representing the `<h1>` element with the text content "Hello, world!".

It's important to note that Babel itself doesn't understand JSX syntax natively; instead, it relies on plugins like `@babel/preset-react` to handle JSX transformation. This preset includes the necessary plugins to transform JSX into JavaScript that can be understood by browsers or other JavaScript environments.

App.js:

```
-----
import "./App.css";
function App() {
  const title = "Welcome to the new blog!";
  const likes = 50;
  // can output: number, string, and array
  // can't output: boolean and objects
  const link = "https://www.google.com/";

  const ninja = { name: "Yoshi", age: 30 };
  // returning dynamic values
  return (
    <div className="App">
      <div className="content">
        <h1>{title}</h1>
        <p>Liked {likes} times</p>
        { /* <p>{ninja}</p> */ }
      </div>
    </div>
  );
}
```

```

    <p>`${ninja.name} is a ${ninja.age} years old ninja!`</p>
    { /* bunch different element in arrays together and bunch them as string
*/}

    <p>{[1, 25, 686]}</p>
    { /* use dynamic values as attribute values in element tags */}
    <p>{Math.random() * 20}</p>
    <a rel="noreferrer" target="_blank" href={link}>
      Link to google!
    </a>
  </div>
</div>
);
}

```

```
export default App;
```

Components contain template which makes up the html along with js logic. Components return jsx and In background a transpiler named babel converts all this jsx template into regular JavaScript code when we save the file. class is a reserved keyword in js so we can't use it in jsx.

Components return jsx and exported to use elsewhere.

React converts out (whatever data type we give it to it) into strings before it renders it in the browser.

can output: number, string, and array
 can't output: boolean and objects

Lesson 8(useState hook):

State of a component means data being used on that component in a particular time.

It could be an array of values, booleans, string, objects or any other data that our component uses.

We created some variables and used them in our template before (refer to React Practice.pdf App.js file) but if we need to change variable or data over time or in reaction to some event (user clicking a button)

Home.js (Before using useState Hook):

```
// Destructure useState hook from react library, this grabs us the function
from the react library, now we can use it in our component
```

```

import { useState } from "react";

const Home = () => {
  let name = "Mario";
  const handleClick = () => {
    // Updating the value will not reflect in the template
    name = "Luigi";
    /* name itself changed to luigi but it doesn't get updated in the template
    (view in browser console) cause the variable we have created is not reactive
    (let name = "Mario") means react doesn't watch it for changes, when its value
    changes it doesn't trigger react to re-render the template with the new value
    inside it (name variable in this case) and we continue to see the old value
    (Mario) in the browser, to make this work we need to make the value reactive so
    that when it changes react detects that and it re-renders the template with the
    new value (where we output it in the template) and updated value is visible to
    the browser to implement this we use hook whis is called useState(). Hook in
    react is a special type of function that does a certain job, it generally
    starts with the word use. useState hook provides us a way to make reactive
    value, also provides a way to change that value whenever we want. So to use
    useState hook we need to import it */
    console.log(name);
  };

  // const handleClickAgain = (name, e) => {
  //   console.log(`Hello ${name}!, ${e.target}`);
  // };

  return (
    <div className="home">
      <h2>Homepage</h2>

      <p>{name}</p>
      /* onclicking button change the value of name variable */
      <button onClick={handleClick}>Click me!</button>
      /* <button onClick={(e) => handleClickAgain("Mario", e)}>
        Click me again!
      </button> */
    </div>
  );
};

export default Home;

```

Home.js (After using useState Hook):

```
// Destructure useState hook from react library, this grabs us the function
from the react library, now we can use it in our component
import { useState } from "react";

const Home = () => {
  // let name = "Mario";
  // Make a reactive value using useState, we give this useState a initial
  value for example (Mario), we want to store this in some variable. We use array
  destructuring to grab 2 values that this hook returns to us, first value is the
  initial value (name), 2nd value is a function that we can use to change that
  value most times it's called set[whatever the name of value to be changed]
  const [name, setName] = useState("Mario");
  const [age, setAge] = useState(23);
  const handleClick = () => {
    /* changing the state data using setName, this useState value is reactive
    if we change it it's gonna change in the template as well, when we use this
    function to change the value that triggers react to re-render the component
    upon re-rendering it has the new value of name cause it's been updated. We can
    use this hook (useState) as many times in our component for different values
    other than name like it can be array, object, boolean etc. like 'const [age,
    setAge] = useState(0)' The data type of state we are using doesn't matter */
    setName("Luigi");
    // update age to 30
    setAge(30);
  };

  return (
    <div className="home">
      <h2>Homepage</h2>
      /* At first it will give us the initial value (Mario), to change this
      value we can use the setName function */
      <p>
        {name} is {age} years old.
      </p>
      <button onClick={handleClick}>Click me!</button>
    </div>
  );
};

/* Conclusion: When we need a reactive value something that might change at
some point we use the useState hook to do that we pass in an initial value and
we can output that value in the template and then we just call the set function
2nd value we get in the destructured array to update it and that triggers
re-render and the new value is going to be output to the browser in this
template, so this hook is very useful 😊 */
```

```
export default Home;
```

Lesson 9 (React Dev Tools):

React Dev tools integrate with browser development tools and gives us extra features that we can use on any website created with react. To extra tabs components, and profiler is available. Components is more useful. This gives us component diagram or component tree of our current app. Hovering over it provides extra info about the component.

Important tabs in react dev tools:

1. Inspect the matching DOM element. (eye icon)
2. Log this component data to the console. (bug icon)

Change of state (data) under hooks tab can be viewed in home (by selecting it from component tree) component by clicking the button (click me). Changes in state in a component can be tracked here.

If we log the home component to the console we can see the hooks property with array of objects and each object represents the piece of state we have with properties name, value, id, isStateEditable etc.

Lesson 10 (Outputting Lists):

Goal: Outputting list of blogs in our template.

Create some states to represent these blogs. We will be using useState hook cause data might change at some point we might delete the blog, we need react to update the DOM when that happens.

Home.js (lecture 10: Outputting List)

```
import { useState } from "react";
```

```
const Home = () => {  
  /* destructure the two values, initial value of this state is an array of  
  objects, each objects represent a blog with title, body, author, and id  
  property. This id is going to be used by react when we output this data, each  
  id needs to be unique for each one of the items/blogs. */  
  const [blogs, setBlogs] = useState([  
    { title: "My new website", body: "lorem ipsum...", author: "mario", id: 1  
  },  
    { title: "Welcome party!", body: "lorem ipsum...", author: "yoshi", id: 2  
  },  
    {  
      title: "Web dev top tips",
```

```

        body: "lorem ipsum...",
        author: "mario",
        id: 3,
    },
]);
return (
    <div className="home">
        {/* iterate/loop through blogs array using map method, for each iteration
        as we iterate through this we get access to the item we are currently iterating
        that is blog in this case, when we output list using map method each root
        element in the template that we return must have a key property now this key
        property is something that react uses to keep track of each item in the dom as
        it outputs it, so if data changes at any point say we remove/add new items to
        the array react can keep track of those items. So we must add a key attribute
        to each item that we output, otherwise react can't distinguish between list
        items in the DOM. This normally an id property for each item in the array (we
        already have that in our array of objects) */}
        {blogs.map((blog) => (
            // what we want to output for each blog
            <div className="blog-preview" key={blog.id}>
                <h2>{blog.title}</h2>
                <p>Written by {blog.author}</p>
            </div>
        ))}
    </div>
);
};

```

/* Summary: That is how we output a list of data in react. We have a list in this case an array of object which is stored in useState the we map through the data and we take each item into that as we map through it an we output a bit of template for each one (in this case see comment what we want to output for each blog) and each one has a key property which is id in our case but it could be any unique property. Now some css is added for each blog template (view after 'blog previews / list' comment in index.css) */

```
export default Home;
```

Lesson 11 (Props):

If we are building a real blog we might have the list of blogs in various places on our website it might be the homepage, search, category or tag page. So several different areas may use the same logic where we cycling/iterating through blogs and outputting a blog preview for each one. To implement that in our project we'll be repeating the code (map function iteration) over and over again in different components for different pages. Where we have pieces of

components or bits of templates that might be reused in different positions or different places in the website we like to make that bit of template into its own reusable component. e.g. if we make a component blog list then we could drop this blog list component in any other components in the project. So if we have category page later on we could just get the blog list component and drop it in. To pass in different data into reusable component every time use it we will do that in form of props. E.g. in the homepage we might list all the blogs and show a preview for all the blogs starting from the latest one but on a search page or search component, we might only show the blogs that match the search term so the data is going to be different the structure is the same (of map function iteration BlogList component) but the blogs that we are going to use (state array of objects/blogs) is going to be different, so we can pass in data into these external components as well in the form of props. An external component of BlogList will contain all the logic/template(jsx) of map iteration and listing blog preview.

Home.js

```
-----
import { useState } from "react";
import BlogList from "../BlogList";

const Home = () => {
  const [blogs, setBlogs] = useState([
    { title: "My new website", body: "lorem ipsum...", author: "mario", id: 1 },
    { title: "Welcome party!", body: "lorem ipsum...", author: "yoshi", id: 2 },
    {
      title: "Web dev top tips",
      body: "lorem ipsum...",
      author: "mario",
      id: 3,
    },
  ]);
  return (
    <div className="home">
      { /* {} - Dynamic value */ }
      <BlogList blogs={blogs} title="All Blogs!" />
    </div>
  );
};

/* Conclusion: That's how we can make a component take in props data and then
use that data inside that component. It makes the Bloglist component more
reusable and it does, we can now use this BlogList component anywhere in our
application whether in home component or in different page component later on
*/
```

```
export default Home;
```

```
-----  
BlogList.js  
-----
```

```
// const BlogList = (props) => {  
const BlogList = ({ blogs, title }) => {  
  /* storing blog property of props object in blogs, we are passing in  
properties to props object and grabbing this different properties from this  
props object and storing these in this variables now an easier way to do this  
is destructuring ('{blogs, title}') as we want blogs and title from props  
object */  
  // const blogs = props.blogs;  
  // const title = props.title;  
  // const { blogs, title } = props;  
  // console.log(props, blogs);  
  return (  
    /* In the BlogList component we are trying to map through the blogs data  
but this component has no idea of what blogs is. The blogs data is not defined  
in this component. We can't just use any data in another where it's defined  
(Home component in this case) right here in home component because it can't  
reach that. There is 2 ways to fix this: 1st option is to redeclare all this  
data into BlogList component instead of home (data - ' const [blogs, setBlogs]  
= useState([  
  { title: "My new website", body: "lorem ipsum...", author: "mario", id: 1  
},  
  { title: "Welcome party!", body: "lorem ipsum...", author: "yoshi", id: 2  
},  
  {  
    title: "Web dev top tips",  
    body: "lorem ipsum...",  
    author: "mario",  
    id: 3,  
  },  
  ]);' the 2nd option is to use props whereby we pass this data from this home  
component into the BlogList here the 2nd option props will be used for 3  
reasons 1st: It's gonna make our BlogList component more reusable and we'll see  
exactly how later on. 2nd: It allows us to still use this data in home  
component later on (cause we are not removing it) if we need it in future cause  
the data is still gonna be declared here in home component. 3rd: It allows to  
learn how to use props 😊. Props are a way to pass data from one component( a  
parent component ) into a child component. In this case Home is parent  
component and BlogList is child component, we will be passing the blogs data  
into the BlogList component to do that we will make a property name on  
<BlogList/> tag. We can call it whatever we want it's blogs in this case. So,  
<BlogList blogs={blogs} /> now this is being passed into BlogList component (  
check home component code ) as a prop ( blogs={blogs} - this is a prop ). We
```

need to receive it here in BlogList component we get access to an argument inside this function/component called props. Now this blogs property will be in props object. Any props that we send through (like blogs={blogs}) into a component will be attached to this props object which we automatically get as an argument in the component ('const BlogList = (props)=>{') and we can access them like props.blogs. We can pass in multiple props if we want to like we can pass in title="All Blogs" with a string value */

```
<div className="blog-list">
  <h2>{title}</h2>
  {/* {props.blogs.map((blog) => ( */}
  {blogs.map((blog) => (
    // what we want to output for each blog
    <div className="blog-preview" key={blog.id}>
      <h2>{blog.title}</h2>
      <p>Written by {blog.author}</p>
    </div>
  )})}
</div>
);
};

export default BlogList;
```
