React.js

Regards: Hassan Bilal

TABLE OF CONTENTS

01

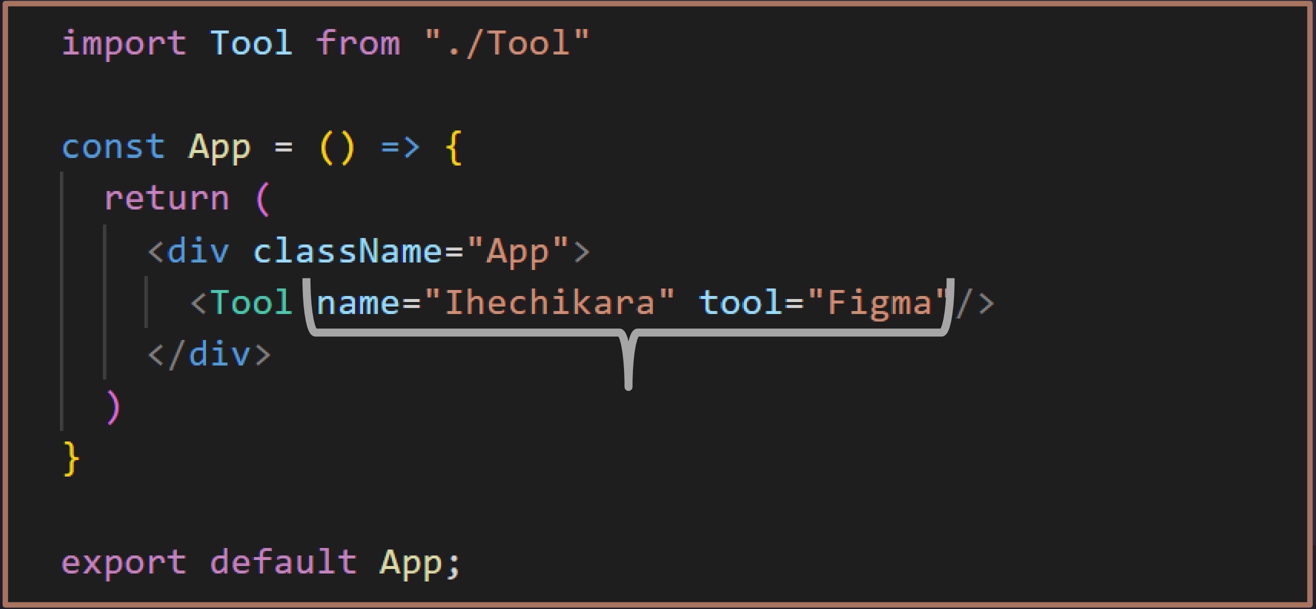
Props

What is Props

- Shorter way of saying properties.
- To pass data from one component to another.
- From a parent component to a child component(s).
- They are useful when you want the flow of data in your app to be dynamic.

Props – Example

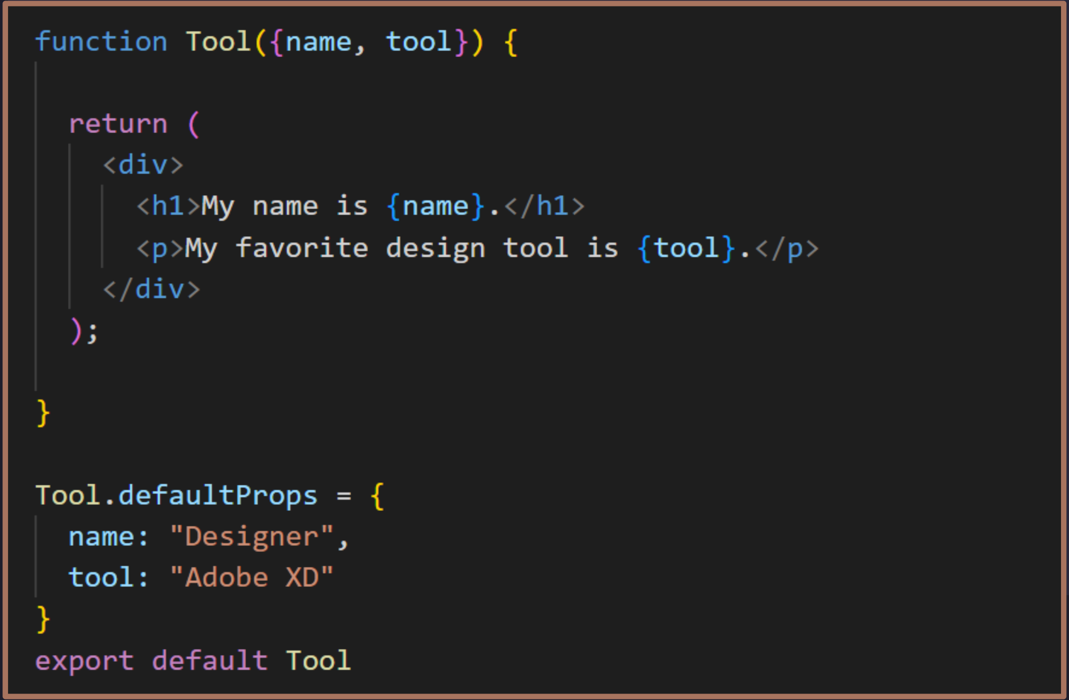
- name and tool will be passed down to the Tool component as props.



```
import Tool from "./Tool"

const App = () => {
  return (
    <div className="App">
      <Tool name="Ihechikara" tool="Figma"/>
    </div>
  )
}

export default App;
```

Props – Example

- name and tool will be received inside the components in props.



```
function Tool({name, tool}) {

  return (
    <div>
      <h1>My name is {name}.</h1>
      <p>My favorite design tool is {tool}.</p>
    </div>
  );

}

Tool.defaultProps = {
  name: "Designer",
  tool: "Adobe XD"
}
export default Tool
```

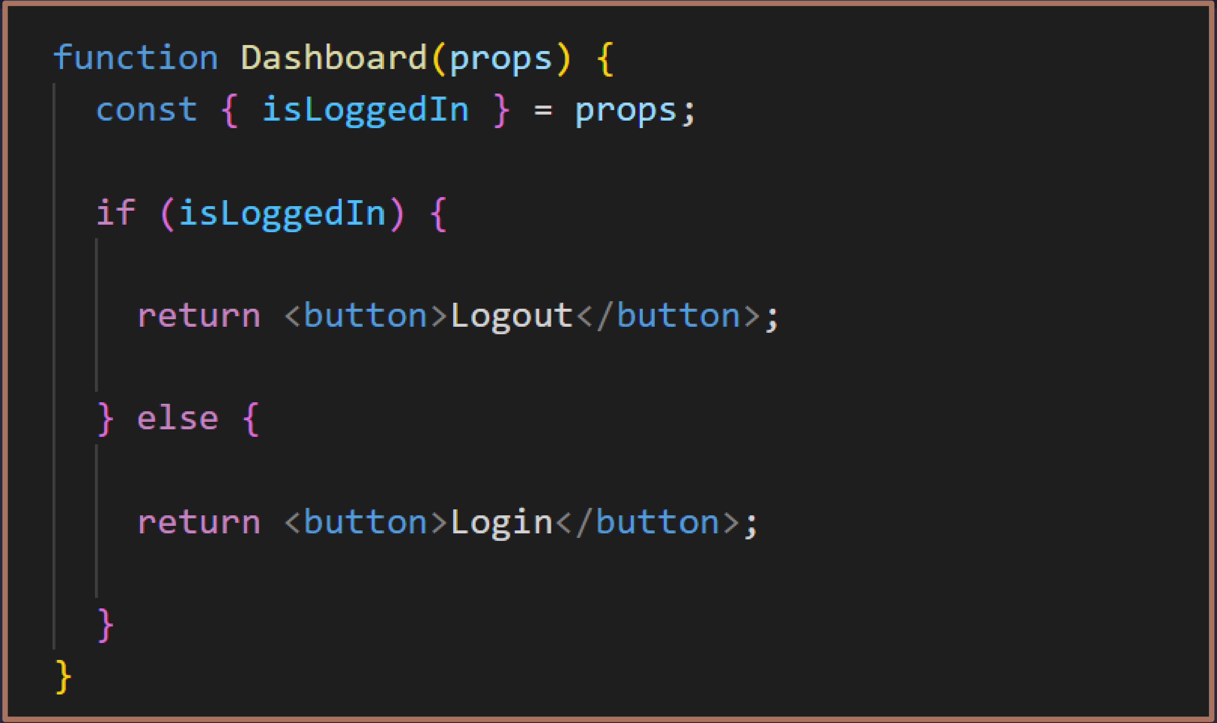
01

Conditional rendering

What is Conditional rendering

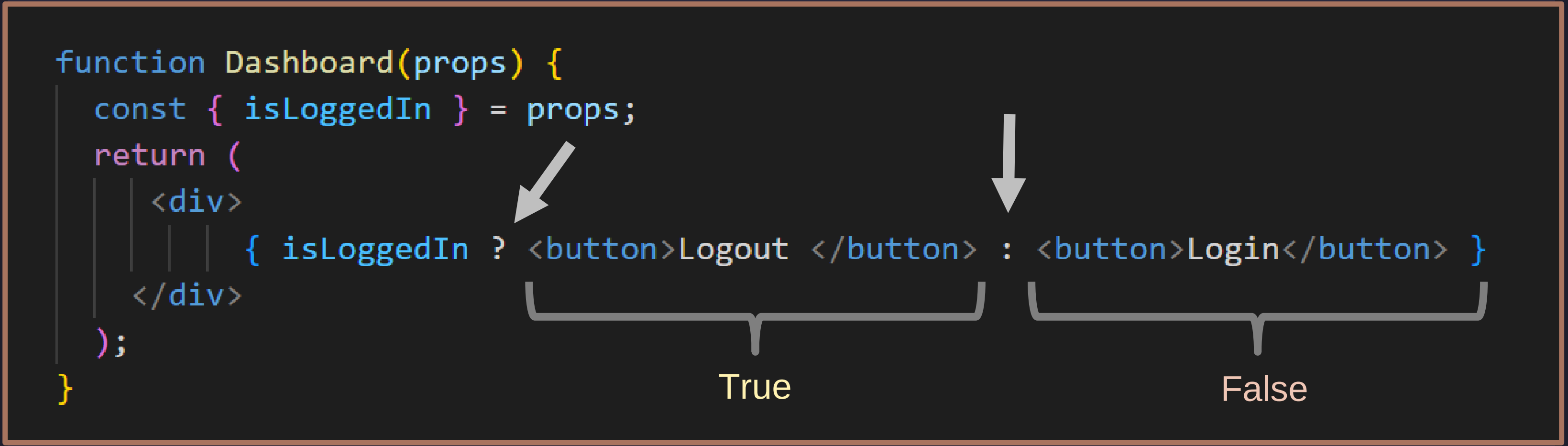
- Process of delivering elements and components based on certain conditions.
- There’s more than one way to use conditional rendering in React.

Conditional rendering – if else



```
function Dashboard(props) {
  const { isLoggedIn } = props;

  if (isLoggedIn) {

    return <button>Logout</button>;

  } else {

    return <button>Login</button>;

  }
}
```

Conditional rendering – ternary operator

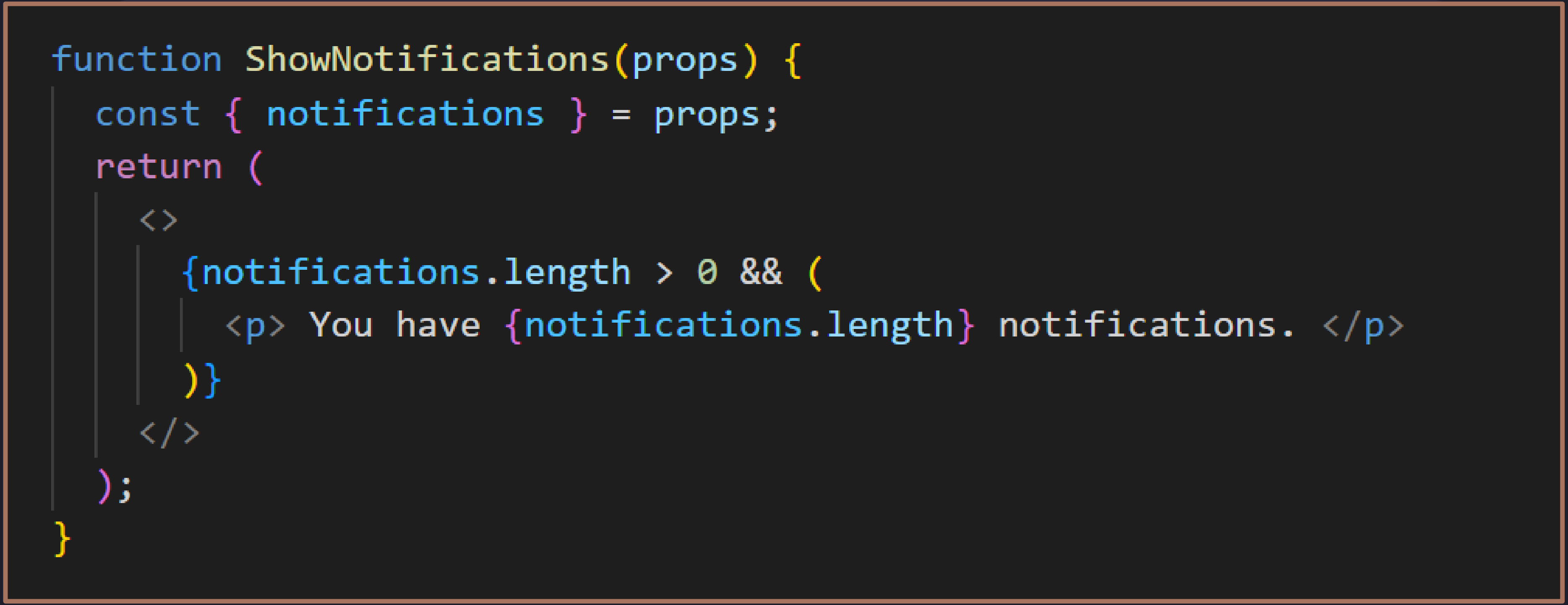


```
function Dashboard(props) {
  const { isLoggedIn } = props;
  return (
    <div>
      { isLoggedIn ? <button>Logout </button> : <button>Login</button> }
    </div>
  );
}
```

Conditional rendering – element variable

```jsx
function Dashboard(props) {
  const {isLoggedIn} = props;

  let elementVariable;

  if (isLoggedIn) {
    elementVariable = <button> Logout </button>;
  }
  else{
    elementVariable = <button> Login </button>;
  }

  return (
    <>
      {elementVariable}
    </>
  );
}
```

Conditional rendering – logical operator &&



```
function ShowNotifications(props) {
  const { notifications } = props;
  return (
    <>
      {notifications.length > 0 && (
        <p> You have {notifications.length} notifications. </p>
      )}
    </>
  );
}
```

Conditional rendering – Prevent rendering

```
function Warning(props) {
  const { warningMessage } = props;

  if (!warningMessage) {
    return null;
  }
  return (
    <>
      <button>This is some warning text!</button>
    </>
  );
}
```

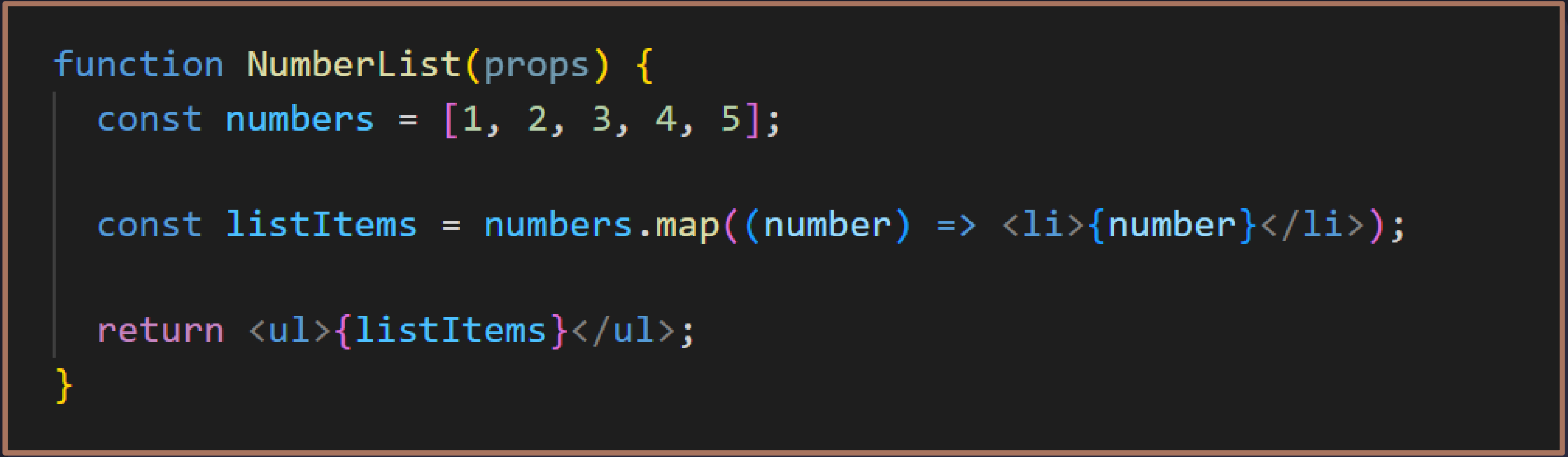
Conditional rendering – Real project example

```
function FetchData() {
  const [data, setData] = useState(null);
  const apiURL = "https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY";

  const fetchData = async () => {
    const response = await fetch(apiURL);
    setData(response.data);
  };

  return (
    <div>
      <h1>Astronomy picture of the day</h1>
      {data && (
        <>
          <p>{data.title}</p>
          <p>{data.explanation}</p>
        </>
      )}
    </div>
  );
}
```

03

Lists and Keys

Lists and keys



```
function NumberList(props) {
  const numbers = [1, 2, 3, 4, 5];

  const listItems = numbers.map((number) => <li>{number}</li>);

  return <ul>{listItems}</ul>;
}
```

- When you run this code, you’ll be given a warning that a key should be provided for list items.

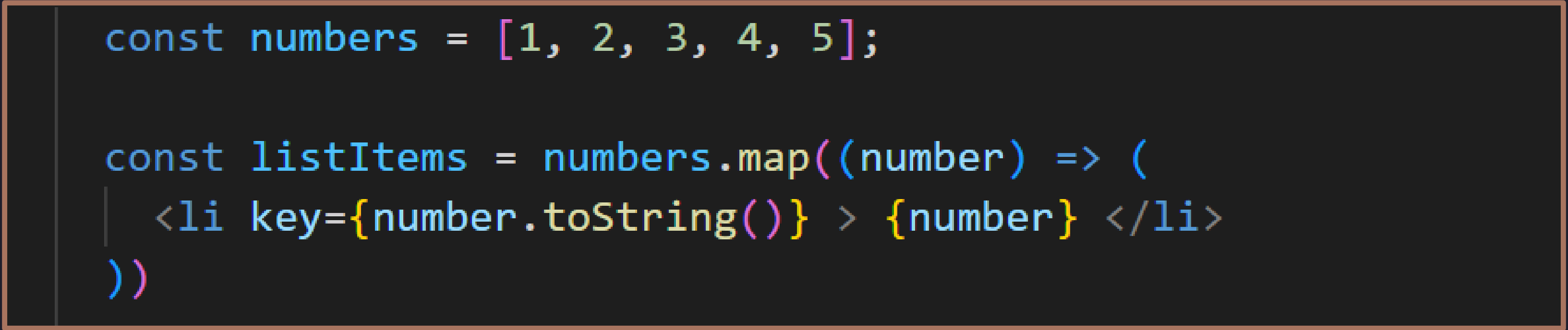
- A “key” is a special string attribute you need to include when creating lists of elements.

Lists and keys

```
function NumberList(props) {
  const numbers = [1, 2, 3, 4, 5];

  const listItems = numbers.map((number) => (
    <li key={number.toString()} > {number} </li>
  ))

  return <ul>{listItems}</ul>;
}
```

Lists and keys – keys – using string as a key

- 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.



```
const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) => (
  <li key={number.toString()} > {number} </li>
))
```

Lists and keys – keys – using ID as a key

- The best way to pick a key is to use a string.
- That uniquely identifies a list item among its siblings.
- Most often you would use IDs from your data as keys:

```
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
);
```

Lists and keys – keys – using index as a key

- When you don’t have stable IDs for rendered items.
- You may use the item index as a key as a last resort:

```
const todoItems = todos.map((todo, index) =>
  // Only do this if items have no stable IDs
  <li key={index}>
    {todo.text}
  </li>
);
```

Lists and keys – keys – using index as a key

- We don’t recommend using indexes for keys if the order of items may change.
- This can negatively impact performance.
- And may cause issues with component state.

04

Embedding map( ) in JSX

Embedding map( ) in JSX

```jsx
function NumberList(props) {
  const numbers = [1, 2, 3, 4, 5]

  return (
    <ul>
      {numbers.map((number) => (
        <ListItem key={number.toString()} value={number} />
      ))}
    </ul>
  );
}
```

<QnA>

> Thanks! <

Regards: Hassan Bilal