

# WORKING WITH LISTS AND CONDITIONALS

---

# Working with lists

```
const SimpleListComponent = () => {  
  const nums = [1,2,3,4,5]  
  const updatedNums = nums.map((num)=>{  
    return <li>{num*num}</li>;  
  });  
  
  const items = [{name:'Item-1'}, {name:'Item-2'}]  
  const updatedItems = items.map(item => (  
    <p>{item.name}</p>  
  ))  
  
  const students = [{name:"Anita",rollno:101},  
    {name:"Sunita",rollno:102},  
    {name:"Kavita",rollno:103}]  
  const updatedStudents = students.map(student => (  
    <tr>  
      <td>Name {student.name} </td>  
      <td>Rollno {student.rollno} </td>  
    </tr>  
  ))  
  return(  
    <div>  
      <h2> Numbers List</h2> {updatedNums}  
      <h2> Items List</h2> {updatedItems}  
      <h2> Students List</h2>  
      <table border="1">{updatedStudents}</table>  
    </div>  
  );  
}  
export default SimpleListComponent
```

## Numbers List

- 1
- 4
- 9
- 16
- 25

## Items List

Item-1  
Item-2

## Students List

Name Anita	Rollno 101
Name Sunita	Rollno 102
Name Kavita	Rollno 103

# Working with lists in child component

```
import SimpleListChildComponent from "./SimpleListChildComponent";
```

```
const SimpleListComponent = () => {  
  const nums = [1,2,3,4,5]  
  
  const items = [{name:'Item-1'}, {name:'Item-2'}]  
  
  const students = [{name:"Anita",rollno:101},  
                    {name:"Sunita",rollno:102},  
                    {name:"Kavita",rollno:103}]
```

```
  return(  
    <div>  
      <SimpleListChildComponent  
        nums={nums}  
        items={items}  
        students={students} />  
    </div>  
  );  
}  
export default SimpleListComponent
```

```
const SimpleListChildComponent = (props) => {  
  
  const nums = props.nums;  
  const updatedNums = nums.map(...);  
  const items = props.items  
  const updatedItems = items.map(...)  
  
  const students = props.students;  
  const updatedStudents = students.map(...)  
  
  return(...);  
}  
export default SimpleListChildComponent;
```

# Working with the expenses list

```
const expenses = [  
  { title: 'Groceries', amount: 900, date: new Date(2020, 7, 14)},  
  { title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },  
  { title: 'SofaSet', amount: 25000, date: new Date(2021, 2, 28)}  
];
```

```
return (  
  <div className="App">  
    <h2>Welcome to React!</h2>  
    <ExpenseItem  
      expDate={expenses[0].date}  
      expTitle={expenses[0].title}  
      expAmount={expenses[0].amount}  
    />  
    <ExpenseItem  
      expDate={expenses[1].date}  
      expTitle={expenses[1].title}  
      expAmount={expenses[1].amount}  
    />  
    <ExpenseItem  
      expDate={expenses[2].date}  
      expTitle={expenses[2].title}  
      expAmount={expenses[2].amount}  
    />  
  </div>  
)  
);  
}
```

# Working with the expenses list

```
function App() {
  const expenses = [
    { title: 'Groceries', amount: 900, date: new Date(2020, 7, 14)},
    { title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },
    { title: 'Sofa Set', amount: 25000, date: new Date(2021, 2, 28)}
  ];
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      {expenses.map(expense => {
        return <ExpenseItem
          expDate={expense.date}
          expTitle={expense.title}
          expAmount={expense.amount}
        />
      })}
      {/* <ExpenseItem
        expDate={expenses[0].date}
        expTitle={expenses[0].title}
        expAmount={expenses[0].amount}
      /> ... */}
    </div>
  );
}
```

**export default App;**

```
{expenses.map(expense =>
  (<ExpenseItem
    expDate={expense.date}
    expTitle={expense.title}
    expAmount={expense.amount}
  />))
}
```

August 14 2020	Groceries	Rs 900	Change Title
March 12 2021	New TV	Rs 34000	Change Title
March 28 2021	Sofa Set	Rs 25000	Change Title

# Using stateful lists

```
const DUMMY_EXP = [
  { title: 'Groceries', amount: 900, date: new Date(2020, 7, 14)},
  { title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },
  { title: 'New Sofa Set', amount: 25000, date: new Date(2021, 2, 28)}
];
```

```
function App() {
  const [expenses, setExpenses] = useState(DUMMY_EXP)

  const addExpenseHandler = expense => {
    setExpenses(prevArr => {return [expense, ...prevArr]})
  }
```

```
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <NewExpense onAddExpense={addExpenseHandler} />
```

```
      {expenses.map(expense => (<ExpenseItem
        expDate={expense.date}
        expTitle={expense.title}
        expAmount={expense.amount}
        />))}
```

```
    </div>
  );
}
```

```
export default App;
```

```
const NewExpense = (props) => {const saveE
```

```
const ExpenseForm = (props) => {...const sub
```

The screenshot shows the expense tracker application. At the top, there is a form with a 'Title' input field containing 'Keyboard', an 'Amount' input field containing '1500', and a 'Date' input field showing '12-2021'. Below the form is an 'Add Expense' button. Below the form is a list of expenses. Each expense item shows the date, title, and amount, along with a 'Change Title' button.

Date	Title	Amount	Action
August 14 2020	Groceries	Rs 900	Change Title
March 12 2021	New TV	Rs 34000	Change Title
March 28 2021	New Sofa Set	Rs 25000	Change Title

This screenshot shows a different state of the application. The 'Title' input field now contains 'Keyboard' and the 'Amount' input field contains 'Rs 1500'. The 'Date' input field shows '04 2021'. The 'Add Expense' button is still present. Below the form, the list of expenses is updated to include the new entry.

Date	Title	Amount	Action
December 04 2021	Keyboard	Rs 1500	Change Title
August 14 2020	Groceries	Rs 900	Change Title
March 12 2021	New TV	Rs 34000	Change Title
March 28 2021	New Sofa Set	Rs 25000	Change Title

# Lists and keys

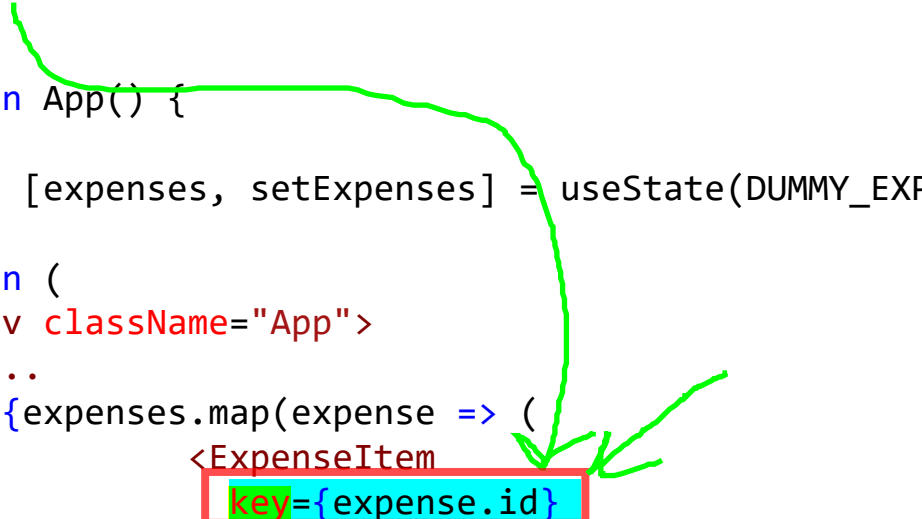
Warning: Each child in a list should have a unique "key" prop. [index.js:1](#)

Check the render method of `App`. See <https://reactjs.org/link/warning-keys> for more information.

at ExpenseItem (<http://localhost:3000/static/js/main.chunk.js:685:19>)  
at App (<http://localhost:3000/static/js/main.chunk.js:184:89>)

```
const DUMMY_EXP = [  
  { id:101, title: 'Groceries', amount: 900, date: new Date(2020, 7, 14)},  
  { id:102, title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },  
  { id:103, title: 'New Sofa Set', amount: 25000, date: new Date(2021, 2, 28)}  
];
```

```
function App() {  
  
  const [expenses, setExpenses] = useState(DUMMY_EXP)  
  ...  
  return (  
    <div className="App">  
      ...  
      {expenses.map(expense => (  
        <ExpenseItem  
          key={expense.id}  
          expDate={expense.date}  
          expTitle={expense.title}  
          expAmount={expense.amount}  
        />))  
      }  
    )  
  )  
  ...  
}
```



# Lists and keys

- When creating a list in JSX, React may show you an error and ask for a key.
  - Keys are unique identifiers that must be attached to the top-level element inside a map.
  - Keys are used by React to know how to update a list whether adding, updating, or deleting items.
  - This is part of how React is so fast with large lists.
  - Keys are a way to help React know how to efficiently update a list.
  - We can add a key using the key prop like so:

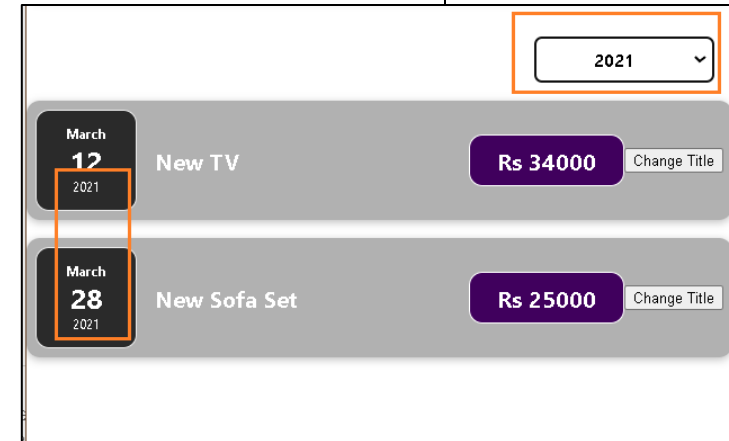
```
<div>
  {people.map(person => (
    <p key={person.name}>{person.name}</p>
  ))}
</div>
```



# Implementing filters

```
function App() {  
  const [expenses, setExpenses] = useState(DUMMY_EXP)  
  const [filteredYear, setFilteredYear] = useState(2020);  
  2021  
  
  const filteredExpensesArr = expenses.filter(expense => {  
    return expense.date.getFullYear().toString() === filteredYear;  
  })  
}
```

```
...  
const selectYearHandler = filteredValue => {  
  setFilteredYear(filteredValue)  
}  
  
return (  
  <div className="App">  
    <h2>Welcome to React!</h2>  
    <NewExpense onAddExpense={addExpenseHandler} />  
    <ExpensesFilter onSelectYear={selectYearHandler}/>  
    {filteredExpensesArr.map(expense => (  
      <ExpenseItem  
        key={expense.id}  
        expDate={expense.date}  
        expTitle={expense.title}  
        expAmount={expense.amount}  
      />))  
  )  
}
```



# Rendering content conditionally

- Conditional rendering means to render a specific HTML element or React component depending on a prop or state value.
  - In a conditional render, a React component decides based on one or several conditions which DOM elements it will return.
  - For instance, based on some logic it can either return a list of items or a text that says "Sorry, the list is empty".

```
if(condition_is_met) {  
    renderSectionOfUI();  
}
```

# Rendering content conditionally : example

```
/*const users = [  
  { id: '1', firstName: 'Shrilata', lastName: 'T' },  
  { id: '2', firstName: 'Anita', lastName: 'Patil' },  
];*/  
const users = []
```

```
function ListUsers() {  
  return (  
    <div>  
      <List list={users} />  
    </div>  
  );  
}
```

```
if (!list.length) {  
  return <p>Sorry, the list is empty.</p>;  
}
```

```
function List({ list }) {  
  if (!list) {  
    return null;  
  }  
  return (  
    <ul>  
      {list.map(item => (  
        <Item key={item.id} item={item} />  
      ))}  
    </ul>  
  );  
}
```

```
function Item({ item }) {  
  return (  
    <li>  
      {item.firstName} {item.lastName}  
    </li>  
  );  
}  
export default ListUsers;
```

## Hello Conditional Rendering

- Shrilata T
- Anita Patil

# Rendering content conditionally : Expense tracker example

```
{filteredExpensesArr.length == 0 ? <p>No expenses found</p> :  
  filteredExpensesArr.map(expense => (  
    <ExpenseItem  
      key={expense.id}  
      expDate={expense.date}  
      expTitle={expense.title}  
      expAmount={expense.amount}  
    />))  
}
```



2022

▼

No expenses found

2020

▼

August

14

2020

Groceries

Rs 900

Change Title

# Rendering content conditionally : one more example

```
const NewExpense = (props) => {  
  const [showForm, setShowForm] = useState(false)  
  
  const showFormHandler = () => {  
    setShowForm(true)  
  }  
  
  const saveExpenseDataHandler = (inputExpenseData) => {...}  
  return(  
    <div className="new-expense">  
      {!showForm && <button onClick={showFormHandler}>Add New Expense </button>}  
      {showForm && <ExpenseForm onCancel={stopShowForm} onSaveExpenseData={saveExpenseDataHandler}/>}  
    </div>  
  );  
}  
export default NewExpense;
```

const stopShowForm = () => {s

```
//ExpenseForm  
<div className="new-expense__actions">  
  <button type="button" onClick={props.onCancel}>Cancel</button>  
  <button type="submit">Add Expense</button>  
</div>
```

The UI mockup illustrates the conditional rendering of the expense form. It features a purple header bar with a dark purple button labeled "Add New Expense". Below this is a grey sidebar containing a calendar for August 14, 2020, and a "Groceries" category. The main content area shows the expense form, which is a purple box with input fields for "Title", "Amount", and "Date" (formatted as dd-mm-yyyy). At the bottom of the form are "Cancel" and "Add Expense" buttons. A date picker is visible below the form, showing "2020". At the bottom of the page, there is a grey footer bar with the same calendar and category, a purple box showing "Rs 900", and a "Change Title" button.