We can use React Fragment to group and display them.

****

OR we can use the other way: <> </>.

****

Each Item of List should have key-property that must be unique in list.

**********p**********

For Conditional Rendering:

```
{ items.length === 0 ? <p>No Items In Jafar-Loka-01 List</p> : null }
```

****

OR we can store the logic inside constant variable:

```
const message = items.length === 0 ? <p>No Items In Jafar-Loka-01 List</p> :
null;
```

```
  return (
    <>
      <h1>Jafar-Loka-01 Test List Group-01</h1>
      { message }
      <ul className="list-group">
        { items.map(item => <li className='list-group-item' key={item}>
{item}</li>)}
      </ul>
    </>
  )
```

****

To create React app using vite: npm create vite.

****

```
const getMessage = () => { return items.length === 0 ? <p>No Items In Jafar-Loka-
01 List</p> : null }
```

****

The benefits of using function are:

- Functions can have parameters.
- Functions can have different brunches.
- Functions can have different logic.
- Functions can have different rendering statements.

****

OR we can use logical operator for rendering.

```
{ items.length === 0 && <p>No Items Found in List</p> }
```

****

Here the result are two expressions will evaluate.

```
{ items.length === 0 ? <p>No Items In Jafar-Loka-01 List</p> : null }
{ items.length === 0 && <p>No Items In Jafar-Loka-01 List</p> }
```

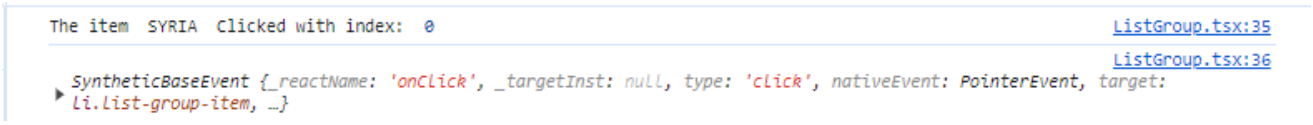
****

For map-function it takes:

- The item.
- The index.

```
{ items.map(
          (item, index) =>
          <li
            className='list-group-item'
            key={item}
            onClick={() => console.log("The item ", item, " Clicked with index:
", index) }>
              {item}
          </li>) }
```

****



****

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

```
const handleClick = (event: MouseEvent) => console.log(event);
```

To avoid any errors or warnings from Hooks-functions we can use:

```
const [ selectedIndex, setSelectedIndex ] = useState<Number>(-1);
```

****

Using React Interface we can define the shape of object, array, interfaces, ...etc. in React.

****

We can use Object Destruction to prevent Code Duplication.

****

```
interface Props {
  items: string[];
  heading: string;
}
```

****

```
const ListGroup = ({ items, heading }: Props) => { ... }
```

```
return (
    <><ListGroup items={items} heading="Jafar-Loka-01 Test List Group-01"/></>
);
```

****

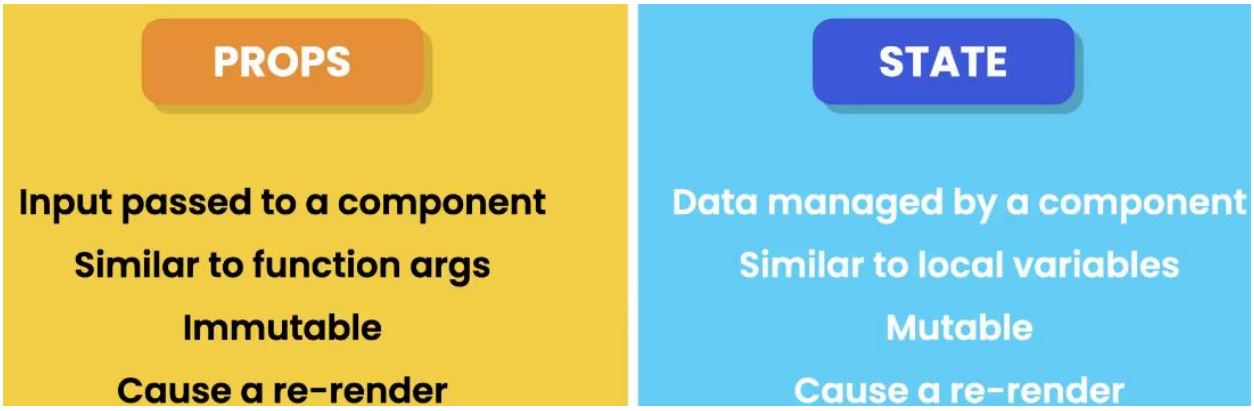
To declare function in interface shape:

```
interface Props {
  items: string[];
  heading: string;
  onSelectedItem: (item: string) => void;
}
```

****

```
const ListGroup = ({ items, heading, onSelectedItem }: Props) => {}
```

****



****

To pass a content to component we use interface with children as property.

```
interface Props {
    children: string;
}
```

```
const Alert = ( { children } : Props) => {
  return (
    <div className='alert alert-primary'>{children}</div>
  )
}
```

```
<Alert>
    Jafar Loka-01 Says: Salam Alekoum
</Alert>
```

****

To pass HTML Component as children we must use: ReactNode as children's type.

****

```tsx
import { ReactNode } from "react";
```

```tsx
interface Props {
    children: ReactNode;
}
```

```tsx
<Alert>
        Jafar Loka-01 Says: Salam Alekoum
        <h2>Jafar Loka-01 is ITE Developer</h2>
</Alert>
```

****

To pass Optional Parameter; we append ? to the name of parameter:

```tsx
interface Props {
    children: string;
    color?: string;
    onClick: () => void;
}
```

```tsx
const Button = ( { children, onClick, color = 'primary' } : Props) => {
  return (
    <button className={"btn btn-"+ color} onClick={onClick}>{ children }</button>
  )
}
```

****

Here we get Compiler Error:

```tsx
<Button onClick={() => console.log('Button Clicked')} color="react">
  Jafar Loka Say: Salam Alekoum
</Button>
```

```tsx
interface Props {
    children: string;
    color?: 'primary' | 'secondary' | 'dark' | 'success' | 'warning';
    onClick: () => void;
}
```

****

If we want to reference the main file in folder we can:

- Create index.ts-file.
- Import the main file that we want.
- Export the main file as default.

****

To create a css module file, we can:

- Create our main component.
- Create our main css module file, with name:
  - ***name-here.module.css***.
- Import the css file in main component as:
  - Import styles from '***./name-here.module.css***'
- Then we can declare the className as:

```
<h1>{heading}</h1>
{items.length === 0 && <p>No item found</p>}
<ul className={styles['list-group']}>
```

OR we can use camel case and write the code like:

```
<h1>{heading}</h1>
{items.length === 0 && <p>No item found</p>}
<ul className={styles.listGroup}>
```

****

If we want to pass more than one class to the component using the previous way:

And using join(' ')-function to create separate classes.

```
<h1>{heading}</h1>
{items.length === 0 && <p>No item found</p>}
<ul className={[styles.listGroup, styles.container].join(' ')}>
```

****

****

To use the CSS-IN-JS, we must first install one of the libraries that support it like: styled-components.

```
PS G:\Web\React\react-game-app-01> npm i styled-components
```

****

To make the object of styled-components we can declare the component like:

Here List is React Component.

```
const List = styled.ul`
  list-style: none;
  padding: 0;
`;
```

****

```
interface ListItemProps {
  active: boolean;
}

const ListItem = styled.li<ListItemProps>`
  padding: 5px 0;
`
```

****

```
const ListItem = styled.li<ListItemProps>`
  padding: 5px 0;
  background: ${ props => props.active ? 'blue' : 'none'}
`
```

****

In this way we define the inline styles.

Here we use CamelCase for defining the styles.

```
<ul className="list-group" style={{ backgroundColor: 'yellow'}}
```

****

To add icons to react app we can use react-icons Library.

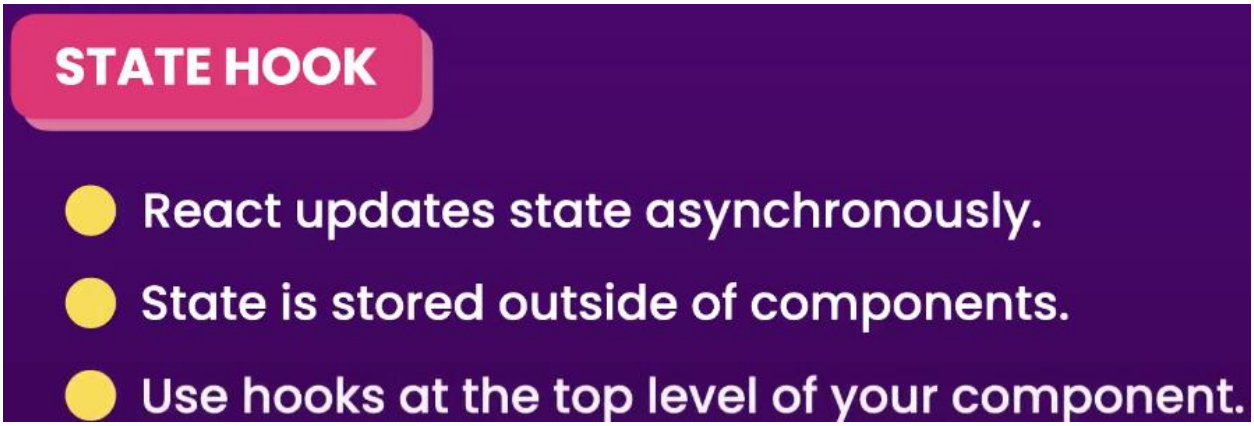
We can install it using npm: ***npm i react-icons***.

****

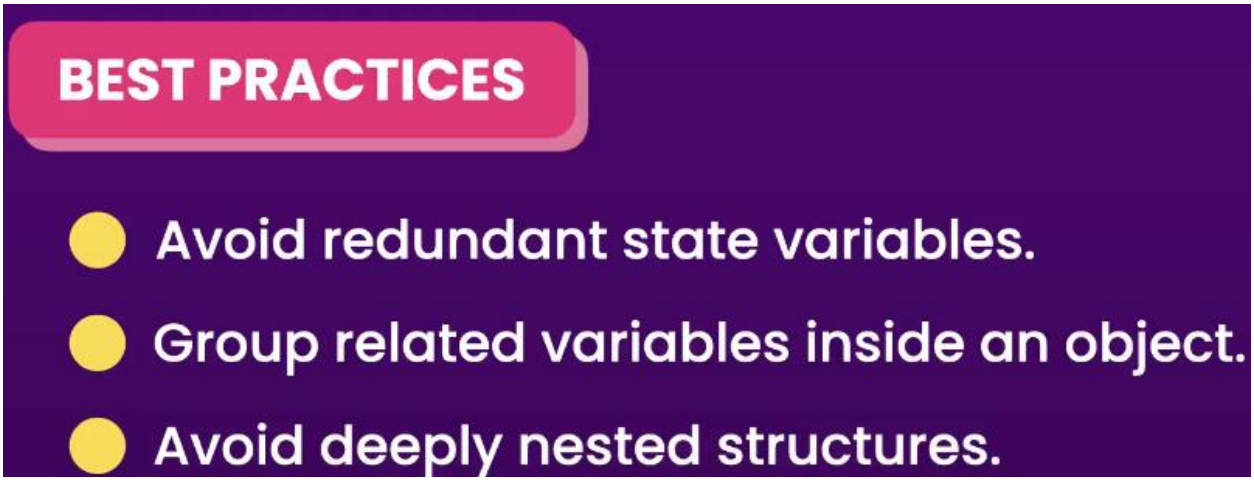
Handle The state management in React is Asynchronous.

This mean if we log the value after call set-function; the old value will be printed.

****


STATE HOOK
React updates state asynchronously.
State is stored outside of components.
Use hooks at the top level of your component.

****


BEST PRACTICES
Avoid redundant state variables.
Group related variables inside an object.
Avoid deeply nested structures.

****


Pure Function
Given the same input, always returns the same result.
const result = myFunc(1);

****

The Spread Operator in JS is shallow; it means when destructing an object, it will return the address of the object in memory.

```
const handleClick = () => {
  setCustomer({
    ...customer,
    address: { ...customer.address, zipCode: 94112 },
  });
};
```

****

Handle Arrays with state management:

```
const handleClick = () => {
  // Add
  setTags([ ...tags, 'exciting']);

  // Remove
  setTags(tags.filter(tag => tag !== 'happy'));

  // Update
  setTags(tags.map(tag => tag === 'happy' ? 'happiness' : tag))
};
```

****

Handle Array of Objects:

```
(bugs.map(bug => bug.id === 1 ? { ...bug, fixed: true } : bug))
```

****

We can use Immer-library to simplify the update-operation.

****