

# Solution Review: Reverse Sort

Learn to sort in a reverse order in a react app.

## We'll cover the following ^

- Solution
- Exercises:

## Solution #

The initial sort direction works for strings, as well as numeric sorts like the reverse sort for JavaScript numbers that arranges them from high to low. Now we need another state to track whether the sort is reversed or normal, to make it more complex:

```
const List = ({ list, onRemoveItem }) => {  
  const [sort, setSort] = React.useState({  
    sortKey: 'NONE',  
    isReverse: false,  
  });  
  
  ...  
};
```

src/App.js

Next, give the sort handler logic to see if the incoming `sortKey` triggers are a normal or reverse sort. If the `sortKey` is the same as the one in the state, it could be a reverse sort, but only if the sort state wasn't already reversed:

```
const List = ({ list, onRemoveItem }) => {  
  const [sort, setSort] = React.useState({  
    sortKey: 'NONE',  
    isReverse: false,  
  });  
  
  const handleSort = sortKey => {  
    const isReverse = sort.sortKey === sortKey && !sort.isReverse;  
  
    setSort({ sortKey: sortKey, isReverse: isReverse });  
  };  
};
```

```
const sortFunction = SORTS[sort.sortKey];
const sortedList = sortFunction(list);

return (
  ...
);
};
```

src/App.js

Lastly, depending on the new `isReverse` state, apply the sort function from the dictionary with or without the built-in JavaScript reverse method for arrays:

```
const List = ({ list, onRemoveItem }) => {
  const [sort, setSort] = React.useState({
    sortKey: 'NONE',
    isReverse: false,
  });

  const handleSort = sortKey => {
    const isReverse = sort.sortKey === sortKey && !sort.isReverse;
    setSort({ sortKey, isReverse });
  };

  const sortFunction = SORTS[sort.sortKey];

  const sortedList = sort.isReverse
    ? sortFunction(list).reverse()
    : sortFunction(list);

  return (
    ...
  );
};
```

src/App.js

The reverse sort is now operational. For the object passed to the state updater function, we use what is called a **shorthand object initializer notation**:

```
const firstName = 'Robin';

const user = {
  firstName: firstName,
};

console.log(user);
// { firstName: "Robin" }
```

src/App.js

When the property name in your object is the same as your variable name, you can omit the key/value pair and just write the name:

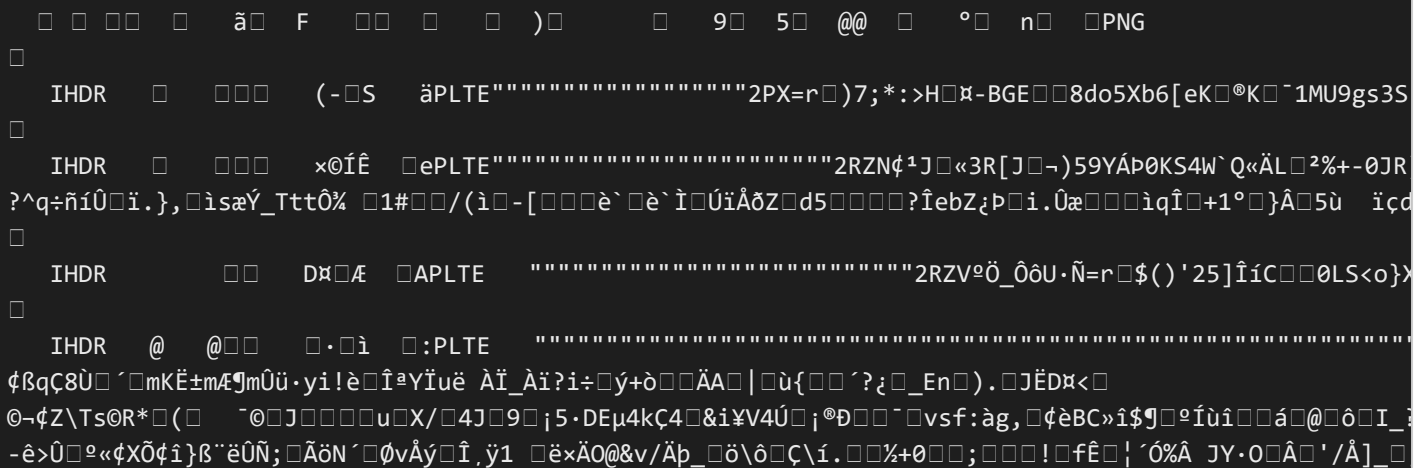
```
const firstName = 'Robin';

const user = {
  firstName,
};

console.log(user);
// { firstName: "Robin" }
```

src/App.js

Here is the complete demonstration of the above concepts:



If necessary, read more about [JavaScript Object Initializers](#).

## Exercises: #

- Confirm the [changes from the last section](#).
- Consider the drawback of keeping the sort state in the List instead of the App component. If you don't know, sort the list by "Title" and search for other stories afterward. What would be different if the sort state would be in the App component?
- Use your styling skills to give the user feedback about the current active sort and its reverse state. It could be an [arrow up or arrow down SVG](#) next to each active sort button.