# Working with lists

map vs FlatList vs SectionList

# Working with lists

## Basics

```
const people = [
  <Text>Chris</Text>,
  <Text>Amanda</Text>,
  <Text>Jason</Text>,
  <Text>Jennifer</Text>
]

return (
  <View>
    { people }
  </View>
);
```

# Working with lists

## Map

```
render() {
  const people = ['Chris', 'Amanda', 'Jason', ‘Jennifer'];

  return (
    <ScrollView>
      { people.map(person => <Text>{person}</Text>) }
    </ScrollView>
  );
}
```

# Working with lists

## Map - key

```
render() {
  const people = ['Chris', 'Amanda', 'Jason', ‘Jennifer'];

  return (
    <ScrollView>
      { people.map((person, index) => <Text key={index}>{person}</Text>) }
    </ScrollView>
  );
}
```

# Working with lists

## Map - key - uuid

```
npm install uuid
```

```
const uuidV4 = require('uuid/v4');
```

```
render() {
  const people = ['Chris', 'Amanda', 'Jason', ‘Jennifer'];

  return (
    <ScrollView>
      { people.map(person => <Text key={uuidV4()}>{person}</Text>) }
    </ScrollView>
  );
}
```

# Working with lists

## Map - class method

```
renderPerson = (person, index) => {
  return (
    <View>
      <Text>{person}</Text>
    </View>
  )
}

render() {
  const people = ['Chris', 'Amanda', 'Jason', ‘Jennifer'];

  return (
    <ScrollView>
      { people.map(this.renderPerson) }
    </ScrollView>
  ;
}
```

# Working with lists

## Map - class method

```
renderPerson = (people) => {
  return people.map((person, index) => {
    return (
      <Text key={index}>{person}</Text>
    );
  });
}

render() {
  const people = ['Chris', 'Amanda', 'Jason', 'Jennifer'];
  return (
    <ScrollView>
      { this.renderPerson(people) }
    </ScrollView>
  );
}
```

# Working with lists

## FlatList

At a minimum needs a `data` and `renderItem` prop

```
render() {
  const data = [{ key: 'Chris' }, { key: 'Amanda' }];

  return (
    <View>
      <FlatList
        data={data}
        renderItem={({ item }) => <Text>{item.key}</Text>}
      />
    </View>
  );
}
```

# Working with lists

## FlatList

If there is no key in the data array,  a keyExtractor function must be passed in as a prop.

```
render() {
  const data = [{ name: ‘Chris' }, { name: ‘Amanda' }];

  return (
    <View>
      <FlatList
        data={data}
        renderItem={({ item }) => <Text>{item.name}</Text>}
        keyExtractor={item => item.name}
      />
    </View>
  );
}
```

# Working with lists

## FlatList

`renderItem` can also be, and usually is, moved into a class method

```
renderItem = ({ item }) => {
  return <Text>{item.name}</Text>
}

render() {
  const data = [{ name: 'Chris' }, { name: 'Amanda' }];

  return (
    <View>
      <FlatList
        data={data}
        renderItem={this.renderItem}
        keyExtractor={item => item.name}
      />
    </View>
  );
}
```

# Working with lists

## FlatList

Also, data is usually either stored as state or received as props.

```
state = {
  data: [{ name: 'Chris' }, { name: 'Amanda' }],
}

renderItem = ({ item }) => {
  return <Text>{item.name}</Text>
}

render() {
  return (
    <View>
      <FlatList
        data={this.state.data}
        renderItem={this.renderItem}
        keyExtractor={item => item.name}
      />
    </View>
  );
}
```

# Working with lists

## FlatList

### Item Separator

```
state = {
  data: [{ name: 'Chris' }, { name: 'Amanda' }],
}

renderItem = ({ item }) => {
  return <Text>{item.name}</Text>
}

render() {
  return (
    <View>
      <FlatList
        data={this.state.data}
        renderItem={this.renderItem}
        keyExtractor={item => item.name}
        ItemSeparatorComponent={() => <View style={styles.divider} />}
      />
    </View>
  );
}
```

# Working with lists

## FlatList

### refreshing prop

```
state = {
  data: [{ name: 'Chris' }, { name: 'Amanda' }],
}

renderItem = ({ item }) => {
  return <Text>{item.name}</Text>
}

render() {
  return (
    <View>
      <FlatList
          data={this.state.data}
          renderItem={this.renderItem}
          keyExtractor={item => item.name}
          refreshing={this.state.refreshing}
          onRefresh={this.onRefresh}
        />
    </View>
  );
}
```

# Working with lists

## FlatList

### Horizontal

```
state = {
  data: [{ name: 'Chris' }, { name: 'Amanda' }],
}

renderItem = ({ item }) => {
  return <Text>{item.name}</Text>
}

render() {
  return (
    <View>
      <FlatList
          data={this.state.data}
          renderItem={this.renderItem}
          keyExtractor={item => item.name}
          horizontal
      />
    </View>
  );
}
```

# Working with lists

## SectionList

At a minimum needs a `sections`, `renderSectionHeader`, and `renderItem` prop

```
const data = [
  {data: [{ name: 'Chris' }], key: 'Basketball'},
  {data: [{ name: 'Amanda' }], key: 'Baseball'},
  {data: [{ name: 'Jennifer' }, { name: 'Mike' }], key: 'Football'},
];

return (
  <View style={styles.container}>
    <SectionList
      renderItem={({item}) => <Text>{item.name}</Text>}
      renderSectionHeader={(section) => {
        return <Text style={styles.header}>{section.section.key}</Text>
      }}
      sections={data}
    />
  </View>
);
```

# Working with lists

## SectionList

Sticky Section Headers. Enabled by default.

```
const data = [
  {data: [{ name: 'Chris' }], key: 'Basketball'},
  {data: [{ name: 'Amanda' }], key: 'Baseball'},
  {data: [{ name: 'Jennifer' }, { name: 'Mike' }], key: 'Football'},
];

return (
  <View style={styles.container}>
    <SectionList
      renderItem={({item}) => <Text>{item.name}</Text>}
      renderSectionHeader={(section) => {
        return <Text style={styles.header}>{section.section.key}</Text>
      }}
      sections={data}
      stickySectionHeadersEnabled={false}
    />
  </View>
);
```