Framework Training
React
London July 2018
Exercise G-basket

- This exercise passes **functions** to components as props. This allows components to invoke functions in their parents.

Installation

- Install and run the starter version of the project.

```
npm install
npm start
```

Review the starter version of the project

- The fruitVeg array is passed into the main component Shop.

```
<Shop fruit={fruitVeg} />
```

- The Shop component render method uses map to iterate over the array.

```
let {fruit} = this.props;
{fruit.map((item,n) => <Item key={n} name={item} /> )}
```

Add items to a basket

- We want to select items and add them to a basket.
- Define an empty basket array as component **state** in the **constructor**.

```
        this.state = { basket:[] };
```

- We can define a buyItem method in the Shop component and pass this down as a prop to the Item component.

```
        buyItem = e => console.log(e);

        <Item key={n} name={item} select={this.buyItem}/>
```

- Inside the Item component, we can add an event handler which points at the select prop.

```
        onClick={props.select}
```

- Clicking on an item will log a React **synthetic event** to the browser console.
- Expression e.target points at the DOM element clicked on.

```
buyItem = e => console.log(e.target);
```

- Store the name of the selected item to a variable.

```
let item = e.target.textContent;
```

- We want to add this name to the basket, but we should not attempt to directly change state.
- This code makes a copy of the basket array using the spread operator and adds in the new item.

```
let item = e.target.textContent;
let copy = [ ...this.state.basket, item ];
```

- We can sort the array of item names.

```
copy = copy.sort( (a,b) => a > b);
```

- Use setState to update the basket.

```
this.setState( { basket:copy } )
```

- Look at the basket state in the React DevTools.

Display the state basket

- Add another FlexBox in the Shop component render method to iterate over the basket array.

```
let {basket} = this.state;

<section className="shelf">
        {basket.map((item,n) => <Item key={n} name={item}/> )}
</section>
```

- This creates a problem. A React render method should return only one top-level element.
- Restructure the JSX: wrap both FlexBoxes in a containing section.
- *Note: both the fruit and basket arrays share the same Item component.*

Conditional styling

- To style the basket items differently, we could create a new component, or use conditional styling.
- This example uses conditional styling.
- Add a type prop to both instances of the Item component

```
<Item .. type="shelf">
<Item .. type="basket">
```

- Use that prop to apply conditional styling.

```
className={ props.type === "basket" ? "basket" : null }
```

Remove items from the basket

- We want to be able to remove items from the basket.
- Define a new method in the main component.

```
removeItem = e => console.log( e.target.textContent );
```

- Pass that function down as a prop.

```
<Item .... select={this.removeItem} />
```

- RemoteItem needs to make a copy of the basket

```
let copy = [ ...this.state.basket ];
```

- Then search the basket for the first element with this name

```
let position = copy.findIndex( (f) => f=== name );
```

- Remove that element from the copy.

```
copy.splice( position,1 );
```

- Use setState to update the state basket.

```
this.setState( { basket : copy } );
```

- Test this: clicking an item in the basket should remove it.