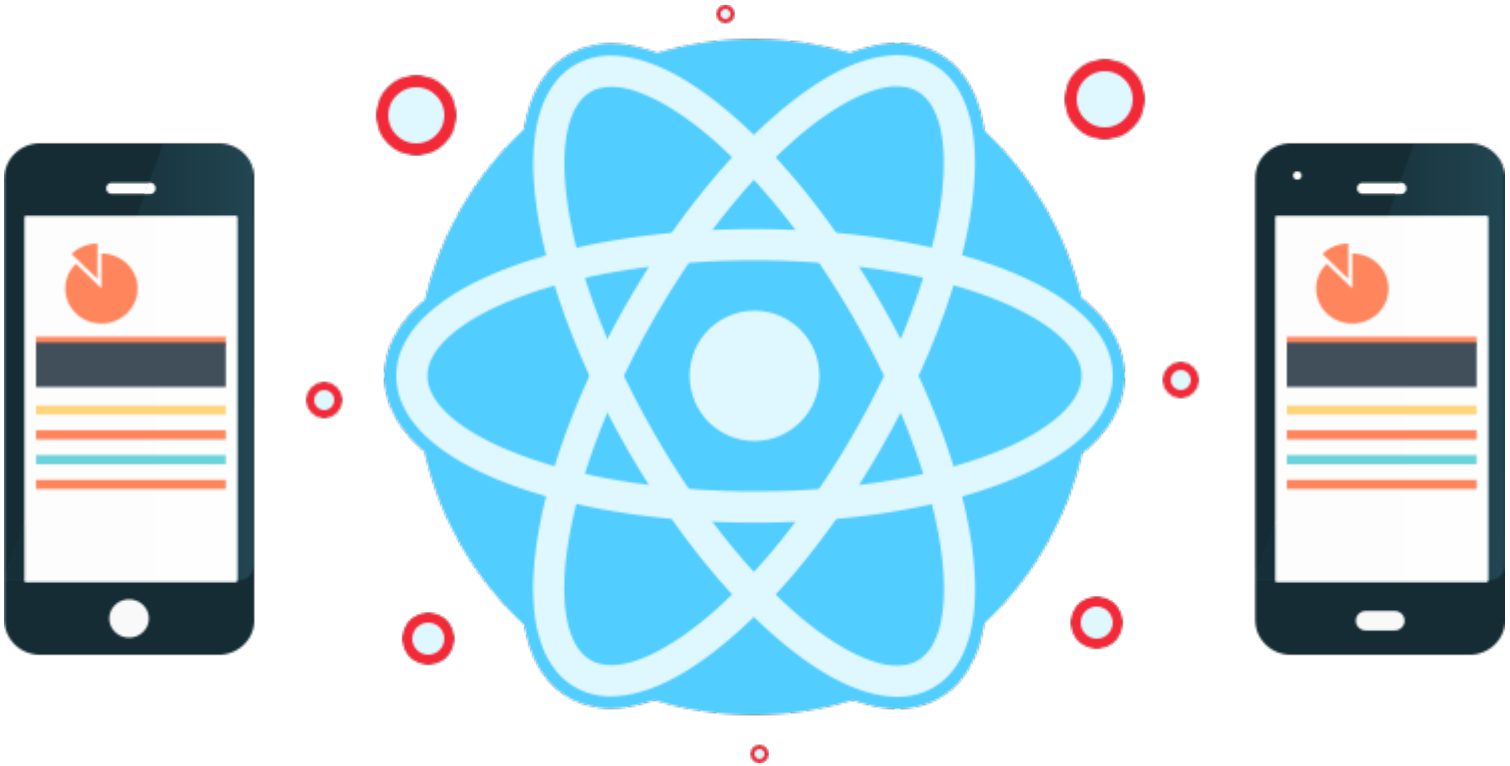
CS385 Mobile Application Development (Lecture 6)



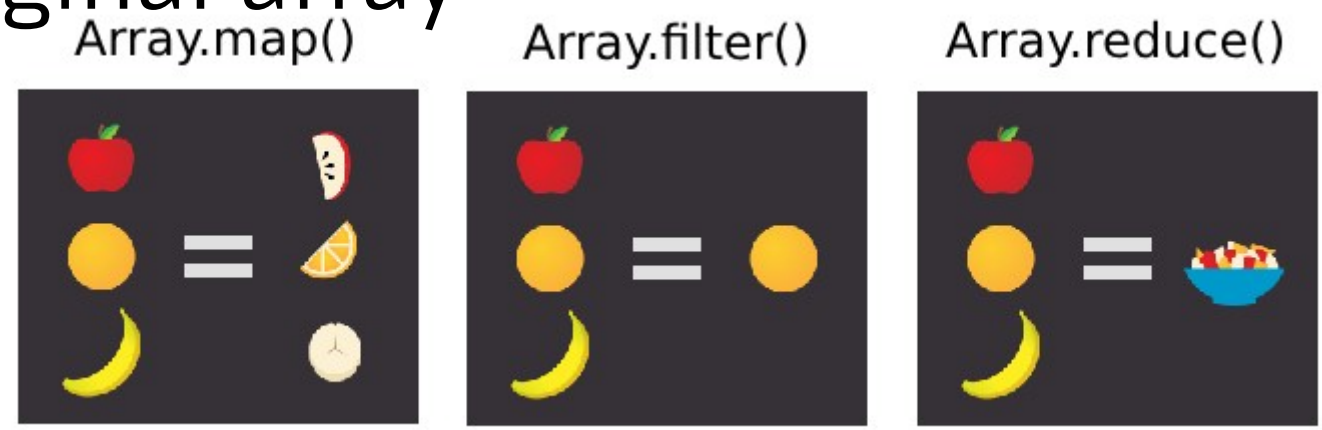
Peter Mooney

The Javascript reduce() function

- Up to this point in CS385 we've seen some very important and powerful Javascript functions namely: the **map** function and the **filter** function.
- There is another very powerful function called the **reduce** function.
- The reduce function also operates on arrays of Javascript objects (just like map and filter)
- Therefore, we can apply it relatively easily

What is the reduce() method?

- The `reduce()` method reduces the array to a single value. The **reduce()** method executes a provided function for each value of the array (from left-to-right). The return value of the function is stored in an accumulator (result/total). This method does not change the original array



A reduce() example

- Suppose we have a simple React application which contains an array of Javascript objects.
- Suppose these object represents purchases in the user shopping basket in online office supply shop.
- We know how to print/render and filter this array. But how can we easily add up the total cost of all of the objects/items in the basket?
- We use the **reduce() function** from Javascript.

A reduce() example (1)

```
function App() {
  let basket = [
    { item: "A4 Paper", price: 10.5 },
    { item: "Stapler", price: 10.0 },
    { item: "Folders", price: 20.5 },
    { item: "Calculator", price: 15.0 }
  ];
  return (
    <>
      Your Shopping Basket:
      {basket.map((b, ind) => (
        <p key={ind}>
          <b>{b.item}</b>, €{b.price}
        </p>
      ))}
    </>
  ); // end of return statement
}
```

Your Shopping Basket:

A4 Paper, €10.5

Stapler, €10

Folders, €20.5

Calculator, €15

We are very familiar with this example at this stage – we are rendering an array with the map function.

A reduce() example (2)- we display the total cost of items

```
function getBasketTotal(acc, obj) {
  return acc + obj.price;
}

function App() {
  let basket = [
    { item: "A4 Paper", price: 10.5 },
    { item: "Stapler", price: 10.0 },
    { item: "Folders", price: 20.5 },
    { item: "Calculator", price: 15.0 }
  ];
  return (
    <>
      Your Shopping Basket Total (€)
      {basket.reduce(getBasketTotal, 0.0)}
      <br />
      Your Shopping Basket:
      {basket.map((b, ind) => (
        <p key={ind}>
          <b>{b.item}</b>, €{b.price}
          <br />
        </p>
      ))}
    </>
  ); // end of return statement
}
```

Your Shopping Basket Total (€)56
Your Shopping Basket:

A4 Paper, €10.5

Stapler, €10

Folders, €20.5

Calculator, €15

The line `{basket.reduce(getBasketTotal, 0.0)}` is where we call the reduce function. The 0.0 is the initial value of the total (we start at zero).

The `getBasketTotal` is our CALLBACK (just like we had for the filter function

A reduce() example (3)- how getBasketTotal works

```
function getBasketTotal(acc, obj) {
  return acc + obj.price;
}

function App() {
  let basket = [
    { item: "A4 Paper", price: 10.5 },
    { item: "Stapler", price: 10.0 },
    { item: "Folders", price: 20.5 },
    { item: "Calculator", price: 15.0 }
  ];
  return (
    <>
      Your Shopping Basket Total (€)
      {basket.reduce(getBasketTotal, 0.0)}
      <br />
      Your Shopping Basket:
      {basket.map((b, ind) => (
        <p key={ind}>
          <b>{b.item}</b>, €{b.price}
          <br />
        </p>
      ))}
    </>
  ); // end of return statement
}
```

Your Shopping Basket Total (€)56
Your Shopping Basket:

A4 Paper, €10.5

Stapler, €10

Folders, €20.5

Calculator, €15

The line `getBasketTotal` **function works in an iterative way like we seen with our filter function callbacks. We start with the accumulator at 0.0. Then the** `getBasketTotal` **is applied to every object in the array (from left to right) and the obj.price is added to the current value of the accumulator.**

A reduce() example (4) – we can easily add more objects to the array

```
function getBasketTotal(acc, obj) {
  return acc + obj.price;
}

function App() {
  let basket = [
    { item: "A4 Paper", price: 10.5 },
    { item: "Stapler", price: 10.0 },
    { item: "Folders", price: 20.5 },
    { item: "Calculator", price: 15.0 },
    { item: "Office Chair", price: 100.0 },
    { item: "Extension chords", price: 20.0 }
  ];
  return (
    <>
      Your Shopping Basket Total (€)
      {basket.reduce(getBasketTotal, 0.0)}
      <br />
      Your Shopping Basket:
      {basket.map((b, ind) => (
        <p key={ind}>
          <b>{b.item}</b>, €{b.price}
          <br />
        </p>
      ))}
    </>
  ); // end of return statement
```

Your Shopping Basket Total (€)176
Your Shopping Basket:

A4 Paper, €10.5

Stapler, €10

Folders, €20.5

Calculator, €15

Office Chair, €100

Extension chords, €20

A reduce() example (5) – reduce also works on empty arrays!

```
function getBasketTotal(acc, obj) {
  return acc + obj.price;
}

function App() {
  let basket = [

  ];
  return (
    <>
      Your Shopping Basket Total (€)
      {basket.reduce(getBasketTotal, 0.0)}
      <br />
      Your Shopping Basket:
      {basket.map((b, ind) => (
        <p key={ind}>
          <b>{b.item}</b>, €{b.price}
          <br />
        </p>
      ))}
    </>
  ); // end of return statement
}
```

Your Shopping Basket Total (€)0
Your Shopping Bask

Let's look at another example of reduce() - again adding the value of a particular property of objects in an array.

This time we are going to use it together with map and filter!

Here we see reduce used with the callback getTotal. This is standard usage of the reduce function

```
// arr.reduce(callback( accumulator, currentValue[, index[, array]] )
function getTotal(acc, obj) {
  return acc + obj.goals;
}

function App() {
  let scores = [
    { team: "Tottenham", goals: 10 },
    { team: "Chelsea", goals: 12 },
    { team: "Newcastle", goals: 11 },
    { team: "Wolves", goals: 14 }
  ];
  return (
    <>
      Total goals is (with filter with reduce)
      {scores.reduce(getTotal, 0)}
      <br />
      Map Function (with filter)
      {scores.map((s, ind) => (
        <p key={ind}>
          {s.team},{s.goals}
        </p>
      ))}
    </>
  );
}
```

Total goals is (with filter with reduce)47
Map Function (with filter)

Tottenham,10

Chelsea,12

Newcastle,11

Wolves,14

Function composition

- **In Javascript it is possible to chain functions together (function composition) so that we direct the output of one function into another.**
- We have seen this with the map function and the filter function. In this situation the filter function only allows a subset of the object array to be passed to the map function.
- In the next example we will use the filter function and the reduce function together. Then we will just get the total or accumulation of the property of a subset of objects.

```
3   function getTotal(acc, obj) {
4     return acc + obj.goals;
5   }
6
7   function myFilter(goalsNumber) {
8     return function (obj) {
9       return obj.goals >= goalsNumber;
10    };
11  }
12
13  function App() {
14    let scores = [
15      { team: "Tottenham", goals: 10 },{ team: "Chelsea", goals:
16      { team: "Newcastle", goals: 11 },{ team: "Wolves", goals: 1
17    ];
18    return (
19      <>
20        Total goals is (with filter with reduce)
21        {scores.filter(myFilter(12)).reduce(getTotal, 0)}
22        <br />
23        Map Function (with filter)
24        {scores.filter(myFilter(12)).map((s, ind) => (
25          <p key={ind}>
26            {s.team},{s.goals}
27          </p>
28        ))}
29      </>
30    );
31  }
```

Total goals is (with filter with reduce)26
Map Function (with filter)

Chelsea,12

Wolves,14

IMPORTANT: Here we see the output of the filter function on the scores array being passed into the reduce function (remember to read line 21 from left to right).
So we filter the scores array first and then the elements which pass the filter are passed to the reduce function. There are only two elements.

Using reduce – demo

Summary - reduce

- We can see that the reduce function is very powerful.
- We can combine it with the filter function.
- We can now perform some arithmetic operations on our arrays of objects without requiring a for loop.
- **It will be asked in several questions in the Lab Exams**

MAP, FILTER, REDUCE

crash course

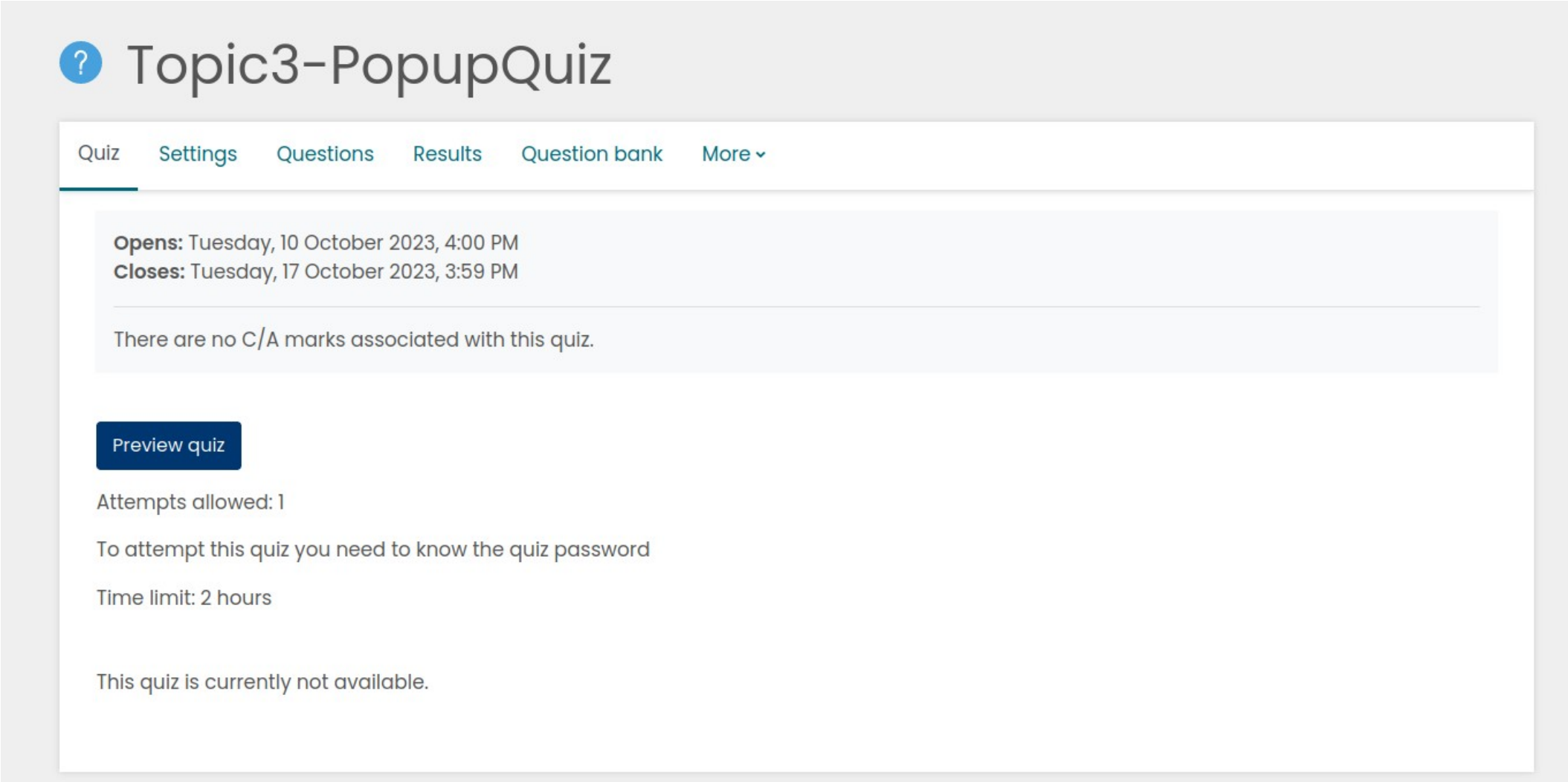
```
[🔥,🚀,🎉]
    .filter(...)
    .map(...)
    .reduce(...)
```

=

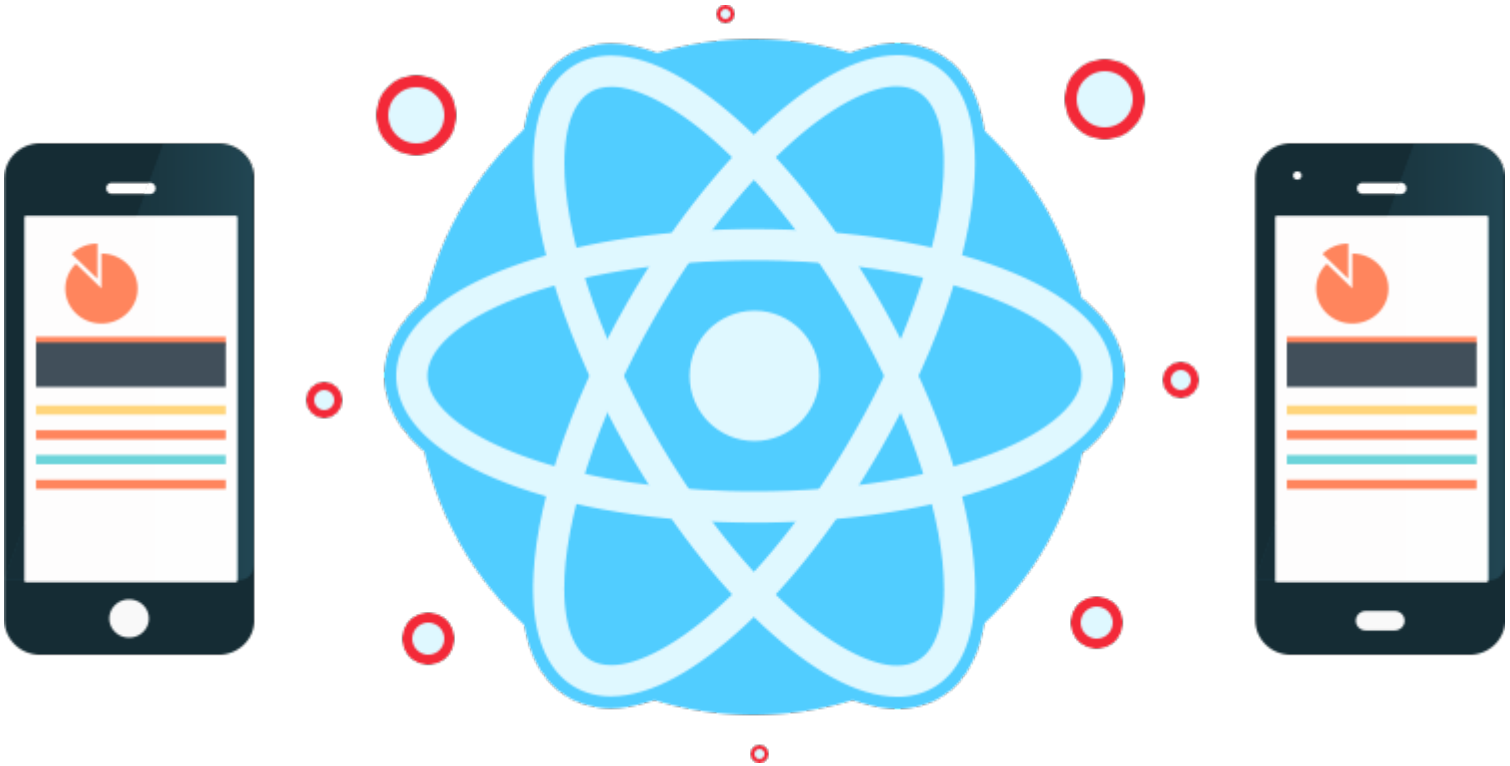
You'll almost never have to write a **for loop** again

Topic 3 – popup quiz

- **This is for your home-work!**



CS385 Mobile Application Development (Lecture 6)



Peter Mooney