COMPSCI 220
Programming Methodology
Exercise: Map, Reduce and Filter

Write a set of functions that takes as input an array of songs as an object:

```
{name: "Example Song", artist: "220 Staff"}
```

And performs the following actions:

1. Return an array of just the song names
2. Return an array of the song names in lower case
3. Return only the songs that are by a given artist
4. Return the number of songs by a given artist

You can access the data with: elem.name and elem.artist
function allSongNames(songs) {
    return songs.map(function(song){
        return song.name;
    });
}
Return an Array of All Song Names Lowercased

```javascript
function allSongNamesLower(songs) {
    return songs.map(function(song){
        return song.name.toLowerCase();
    });
}
```
function songsByArtist(songs, artist) {
    return songs.filter(function(song) {
        return song.artist === artist;
    });
}
Return Number of Songs by a Particular Artist

```javascript
function songCount(songs, artist) {
    return songs.reduce(function(accumulator, song){
        if(song.artist === artist){
            return accumulator += 1;
        }else{
            return accumulator;
        }
    },0);
}
```
Examples of Reduce

**Question**: Using reduce, write a new function called `concat(a1, a2)` that creates a new array consisting of `a2` appended to `a1`
Using reduce for concat

1. What is the type signature of the callback function?
   a. Parameter 1 (current accumulator value)
   b. Parameter 2 (element from array)
   c. Return value (???)

2. How is the accumulator calculated each time?

3. What is the type of initial value, and how do we initialize it?
Using reduce for concat

Assume the input array contains elements of type T

1. What is the type signature of the callback function?
   a. Parameter 1 Array[T]
   b. Parameter 2 (element from array) T
   c. Return value Array[T]

2. How is the accumulator calculated each time?
   Push a new element

3. What is the type of initial value, and how do we initialize it?
   Array[T]: Set to first input array
Exercise: Find the Bug

```javascript
function concat(arr1, arr2) {
    function callback(accumulator, x) {
        accumulator.push(x);
        return accumulator;
    }

    return arr2.reduce(callback, arr1);
}
```
Exercise: Find the Bug

```javascript
function concat(arr1, arr2) {
    function callback(accumulator, x) {
        accumulator.push(x);
        return accumulator;
    }

    return arr2.reduce(callback, arr1);
}
```

Our callback function is calling `push` on the accumulator, that will actually add elements to `arr1` rather than building a completely new array.
function concat(arr1, arr2) {
    function callback(accumulator, x) {
        accumulator.push(x);
        return accumulator;
    }
    return arr2.reduce(callback, arr1.slice());
}
Exercise closestElement

**Question**: Using map and/or reduce, write a new function called `closestElement(array, x)` that returns the element in the input array that is closest in value to `x`.

Both `x` and the elements of array are numbers.

If two elements are equidistant, the earlier element should be returned.

E.g. `closestElement([1, 3, 5, 7, 9], 2)` should return 1.
Using reduce for closestElem

1. What is the type signature of the callback function?
   a. Parameter 1 (current accumulator value)
   b. Parameter 2 (element from array)
   c. Return value (???)
2. How is the accumulator calculated each time?
3. What is the type of initial value, and how do we initialize it?
Using reduce for closestElem

Assume the input Array contains numbers

1. What is the type signature of the callback function?
   a. Parameter 1 Number
   b. Parameter 2 Number
   c. Return value Number

2. How is the accumulator calculated each time?

3. What is the type of initial value, and how do we initialize it?
   Number - First Element in the Array
closestElem

function closestElem(array, x) {
    function callback(acc, elem) {
        if (Math.abs(x - elem) < Math.abs(x - acc)) {
            return elem;
        } else {
            return acc;
        }
    }
    return array.reduce(callback, array[0]);
}
Problem: maxF

Q: Write a HOF that accepts:
An array of functions \([f1, f2, \text{etc.}]\) and number \(x\);
And returns: \(\max(f1(x), f2(x), \text{....})\)
maxF by map and reduce

map: Given \([f_1, f_2, f_3]\)
 produce \([f_1(x), f_2(x), f_3(x)]\)

reduce: Given numbers \([x_1, x_2, x_3]\)
 produce max value of them all
maxF by map and reduce

```javascript
function maxF(fns, x) {
    if (fns.length < 1) {
        return x;
    }
    let values = fns.map(function(f) {
        return f(x);
    });
    let maxValue = fns.reduce(Math.max, values[0]);
    return maxValue;
}
```
maxF with just reduce

1. What is the type signature of the callback function?
   a. Parameter 1 ???
   b. Parameter 2 ???
   c. Return value ???

2. How is the accumulator calculated each time?

3. What is the type of initial value, and how do we initialize it? ???
maxF with just reduce

1. What is the type signature of the callback function?
   a. Parameter 1 Number
   b. Parameter 2 Function(Number) -> Number
   c. Return value Number

2. How is the accumulator calculated each time?

3. What is the type of initial value, and how do we initialize it?
   Number - First Element in the Array
Exercise maxF with just Reduce

```javascript
function maxF(fns, x) {
    if (fns.length < 1) {
        return x;
    }
    function callback(acc, f) {
        return Math.max(acc, f(x));
    }
    let maxValue = fns.reduce(callback, fns[0](x));
    return maxValue;
}
```
Recall from Lecture 4 - Problem: maxF

Q: Write a HOF that accepts an array of functions \([f_1, f_2, \text{ etc.}]\), and returns a function \(f(x)\) such that \(f(x) = \max(f_1(x), f_2(x), \text{ etc.})\)
Using reduce for maxF

1. What is the type signature of the callback function?
   a. Parameter 1 ?
   b. Parameter 2 ?
   c. Return value ?
2. What is the type of the initial value?
3. How do we initialize it?
Using reduce for maxF

1. What is the type signature of the callback function?
   a. Parameter 1: A function returning max over fns seen so far
   b. Parameter 2: Single function from array of functions
   c. Return value: A function returning max over all fns

2. What is the type of the initial value?
   A function

3. How do we initialize it?
   Take the first function from the array
function maxF(fns) {
    if (fns.length < 1) {
        return function(x) {
            return(x);
        }
    }
    function callback(acc, f) {
        return function(x) {
            Math.max(acc(x), f(x));
        }
    }
    let maxValue = fns.reduce(callback, fns[0]);
    return maxValue;
}
Using reduce for closestElemF

1. What is the type signature of the callback function?
   a. Parameter 1?
   b. Parameter 2?
   c. Return value?

2. What is the type of the initial value?

3. How do we initialize it?
Using reduce for closestElemF

1. What is the type signature of the callback function?
   a. Parameter 1: **A function** returning min over elem seen so far
   b. Parameter 2: **Single number** from array of values
   c. Return value: **A function** returning min over the distances

2. What is the type of the initial value?
   **A function**

3. How do we initialize it?
   **A function that returns the first element in the array**
Using reduce for closestElemF

```javascript
function closestElemF(array) {
    function callback(minSoFar, elem) {
        return function(x) {
            let min = minSoFar(x);
            if (Math.abs(min - x) < Math.abs(elem - x)) {
                return min;
            } else {
                return elem;
            }
        }
    }
    return array.reduce(callback, function(x){return array[0]});
}
```
Exercise: Insertion Sort with Reduce

Pseudo code from Wikipedia:

Iterative

\[
\begin{align*}
i & \leftarrow 1 \\
\text{while} \ i < \text{length}(A) & \\
& \quad x \leftarrow A[i] \\
& \quad j \leftarrow i - 1 \\
& \quad \text{while} \ j \geq 0 \text{ and } A[j] > x \\
& \quad \quad A[j+1] \leftarrow A[j] \\
& \quad \quad j \leftarrow j - 1 \\
& \quad \text{end while} \\
& \quad A[j+1] \leftarrow x \\
& \quad i \leftarrow i + 1 \\
\end{align*}
\]

end while

Recursive (Mostly)

\[
\begin{align*}
\text{function} \ insertionSortR & (\text{array } A, \text{ int } n) \\
& \quad \text{if } n > 0 \\
& \quad \quad insertionSortR(A, n-1) \\
& \quad \quad x \leftarrow A[n] \\
& \quad \quad j \leftarrow n-1 \\
& \quad \quad \text{while} \ j \geq 0 \text{ and } A[j] > x \\
& \quad \quad \quad A[j+1] \leftarrow A[j] \\
& \quad \quad \quad j \leftarrow j - 1 \\
& \quad \quad \text{end while} \\
& \quad \quad A[j+1] \leftarrow x \\
& \quad \quad \text{end if} \\
\end{align*}
\]

end function