Function Literals in Environment Diagrams and Array's Filter/Map/Reduce Methods

Lecture 21 - Spring 2020

Trace an Environment Diagram

```
01| let main = async () => {
02|     let input = [2, 3, 4];
03|     let result = reduce(input, mul, 1);
04|     print(result);
05| };
06|
07| let mul: Reducer<number, number> = (m, x) => m * x;
08|
09| let reduce = <T,U>(xs:T[], f:Reducer<T,U>, memo: U):U=>{
10|   for (let i = 0; i < xs.length; i++) {
11|     memo = f(memo, xs[i]);
12|   }
13|   return memo;
14| };
15|
16| main();
```

2. "Just for funcies" -- Given the code, draw an environment diagram at the breakpoint. Once drawn, answer the questions on PollEv.com

```
00  interface Funcy {
01      (a: number, b: number): number;
02  }
03
04  export let main = async () => {
05      let a = 16;
06      let b = 2;
07      let c = justF((or, funcies) => or - funcies, b, a);
08      print(c);
09  };
10
11  let justF = (f:Funcy, a:number, b:number):number => {
12      return f(b / a, a);
13  };
14
15  main();
```

Array's `filter`, `map`, and `reduce` Methods

- Arrays have built-in methods

- Among other methods, arrays have three other built-in, higher-order methods:

1. filter
2. map
3. reduce

Array's **filter** Method

- Every array of type **T[]** has a **filter** method.

- The **filter** method has a single parameter: a **Predicate<T>** of the same type **T**

- For example:

```
let a = [-1, 0, 1, 2];
let b = a.filter((x) => x > 0);
print(b); // Prints: 1, 2
```

- Calling the **filter** method on array **a** will return a new array of type **T**.
  The filter method tests all elements in the original array *using the Predicate<T>*.
  Elements that return true will be copied to the returned array.

Array's **map** Method

- Every array of type **T[]** has a **map** method.

- The **map** method has a single parameter: a **Transform<T, U>** of the same type **T**
  - The **map** method will return an array of type **U[]**

- For example:

```
let a = ["one", "two", "three"];
let b = a.map((s) => s.length);
print(b); // Prints: 3, 3, 5
```

- Calling the **map** method on array **a** will return a new array of type **U[]**.
  The map method transforms all elements in the original array *using the Transform<T,U>*.
  All transformed elements are copied to the returned array in the same order.

Array's **reduce** Method

- Every array of type **T[]** has a **reduce** method.

- The **reduce** method has two parameters:
  1. a **Reducer<T, U>** of the same type **T**
  2. An initial **memo** ("memory" accumulator) value of type **U**

- For example:

```
let a = [1, 2, 3];
let b = a.reduce((memo, x) => memo + x, 0);
print(b); // Prints:6
```

- Calling the **reduce** method on array **a** will return a single value of type **U**. Starting with the initial **memo** parameter, it will call the reducer with memo and each element in **a** successively replacing memo's value with the reducer's returned value. The final **memo** value is returned.

Hands-on: filter/map/reduce Pipeline

- Open **`01-game-stats-app.ts`**

1. Assign to the **filtered** variable the result of calling the **filter** with the **games** List and one of **Predicate** functions below:

```
let filtered: Game[] = games.filter(PREDICATE);
```

2. Assign to the **values** variable, the result of calling **map** with the **filtered** List and one of the **Transform** functions below:

```
let values: number[] = filtered.map(TRANSFORM);
```

3. Assign to the result variable, the result of calling **reduce** with the **values** List and one of the **Reducer** functions below (what should the memo be?):

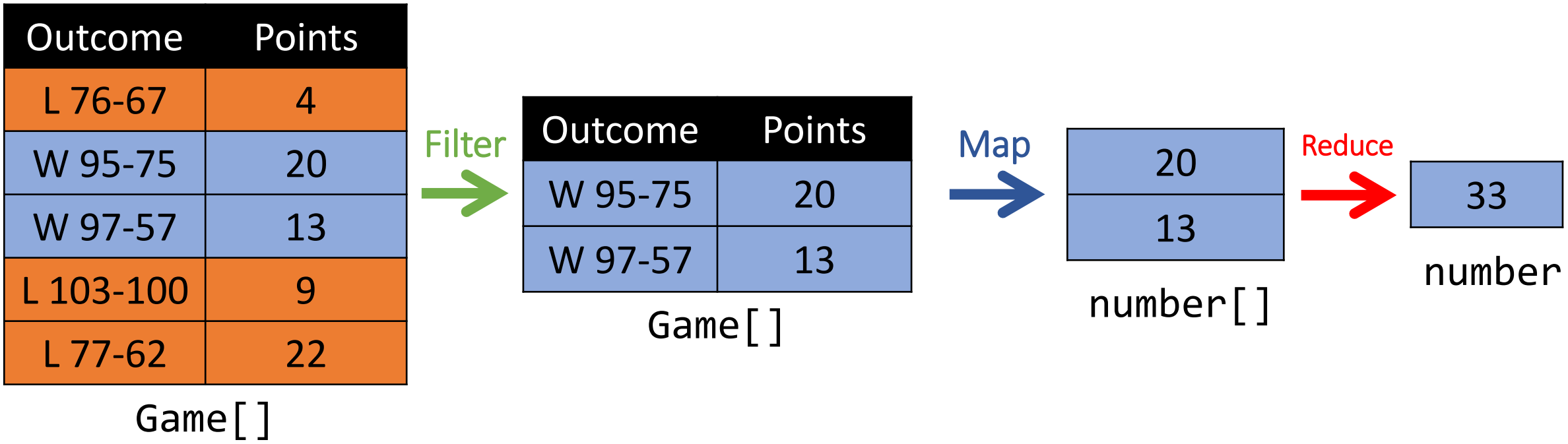
```
let result: number[] = values.reduce(REDUCER, INITIAL_MEMO);
```

4. Now change your code to find the max # of assists Joel Berry had in a game where he scored less than 15 points. Check-in on PollEv.com/compunc when you've got it.

```
// TODO #1
let filtered: Node<Game> = games.filter(fewPoints);
// TODO #2
let values: Node<number> = filtered.map(toAssists);
// TODO #3
let result: number = values.reduce(max, 0);
```

filter-map-reduce Pipeline

Of games that UNC won, how many points did the player score in total?



filter-map-reduce Data Processing Pipeline

Of games

- **that UNC won**
- **that UNC lost**
- **with 3+ assists**
- **with a block**
- **etc**

, what was the

points	**total**
assists	**average**
fouls	**min**
blocks	**max**
etc	**etc**

`Filter:` `Game[]` → `Game[]`

`Map:` `Game[]` → `number[]`

`Reduce:` `number[]` → `number`

Big idea: We can **select any combo of a** filter, map, and reduce sequence.
Result: (# Predicates) x (# Transforms) x (# Reducers) different analyses.

Hands-on: Weather Redux

1. Open 02-weather-redux-app.ts

2. At the first TODO, call the filter method on data. Use an anonymous function as the predicate to filter where a row's precipitation > 0.

3. At the 2nd TODO, assign the total number of rows with precipitation (length of daysWithRain array)

4. At the 3rd TODO, map daysWithRain to a number array with only the precipitation levels of each WeatherRow.

5. At the 4th TODO, reduce to find the sum of precipitation. At the 5th reduce to find the max precipitation