

# Review Session 7

Filter, Map, Reduce

# The Concept

## MAKING SENSE OF DATA

- Pattern for analyzing and processing data
- Data stored in lists/arrays
- Apply functions to make sense of the information
- NOTE: filter, map, and reduce can all be used independently, but they work nicely together in order to find pertinent information from data

# The Concept: Step 1: Filter

- Takes in a collection of data objects (arrays, lists) as argument
- **Returns** a subset of the data objects based on some criteria
- Data type of the parameter ***must match*** the return type

# The Concept: Step 2: Map

- Takes a data set and **converts** it into data of another form
- Use mapping to construct a List of a different type than the one passed in as a parameter
- Useful when you want to focus on a certain property of an object
- Data type of List returned different from the data type of List parameter
  
- Ex: function that takes in a List of words, returns back a List consisting only of the first letters of those words

# The Concept: Step 3: Reduce

- The answer to your question!
- Takes a data set and simplifies it into a final result
  - Result could be a single value OR a collection of values
- Types of the data inputted and type of data returned by the function *must* match

# Example!

- Making sense of today's snow
  - Is this normal?????????

Let's use historical weather data and filter, map, and reduce to see if it is common to have snow at 8:00AM on March 21<sup>st</sup>

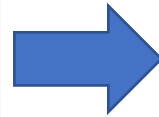
In the past 10 years, how many times has it snowed at 8:00AM on March 21<sup>st</sup>?

year	temp	precipitation
2008	34	false
2009	34	false
2010	48	false
2011	54	true
2012	57	false
2013	35	false
2014	34	false
2015	41	true
2016	38	false
2017	48	true
2018	32	true

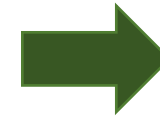
# Using Filter, Map, Reduce to <sup>(not)</sup> Understand Today's Snow

Of all mornings of March 21sts in the past 10 years, how many times did it snow?

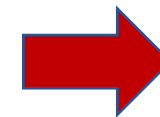
year	temp	precipitation
2008	34	false
2009	34	false
2010	48	false
2011	54	true
2012	57	false
2013	35	false
2014	34	false
2015	41	true
2016	38	false
2017	48	true
2018	32	true



year	temp	precipitation
2011	54	true
2015	41	true
2017	48	true
2018	32	true



54
41
48
32
null



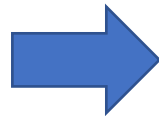
1 number
-------------



# Making Sense of Our Data

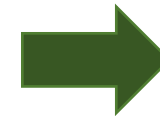
Of all mornings of March 21sts in the past 10 years, how many times did it snow?

year	temp	precipitation
2008	34	false
2009	34	false
2010	48	false
2011	54	true
2012	57	false
2013	35	false
2014	34	false
2015	41	true
2016	38	false
2017	48	true
2018	32	true



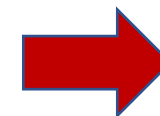
year	temp	precipitation
2011	54	true
2015	41	true
2017	48	true
2018	32	true

Was there any precipitation?



54
41
48
32
null

Was it <= 32 degrees?



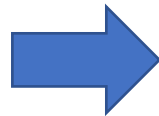
1
number

How many times did this happen?

# Filtering to find precipitation

Of all mornings of March 21sts in the past 10 years, how many times did it snow?

year	temp	precipitation
2008	34	false
2009	34	false
2010	48	false
2011	54	true
2012	57	false
2013	35	false
2014	34	false
2015	41	true
2016	38	false
2017	48	true
2018	32	true



year	temp	precipitation
2011	54	true
2015	41	true
2017	48	true
2018	32	true

Was there any precipitation?

# Filtering to find precipitation

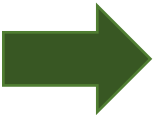
```
let filterByPrecipitating = (list: List<Day>): List<Day> => {  
    if (list === null) {  
        return null;  
    } else if (first(list).precipitation === true) {  
        return cons(first(list), filterByPrecipitating(rest(list)));  
    } else {  
        return filterByPrecipitating(rest(list));  
    }  
};
```

```
let precipitatingDays: List<Day> = filterByPrecipitating(days);
```

# Mapping to find temperatures

Of all mornings of March 21sts in the past 10 years, how many times did it snow?

year	temp	precipitation
2011	54	true
2015	41	true
2017	48	true
2018	32	true



54
41
48
32
null

Was it  $\leq 32$  degrees?

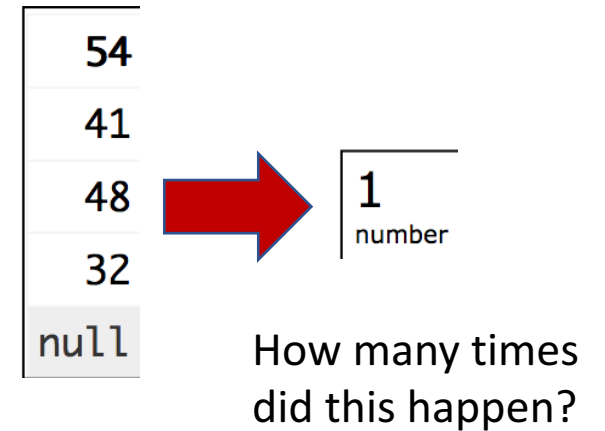
## Mapping to find temperatures

```
let tempify = (day: List<Day>): List<number> => {  
    if (day === null) {  
        return null;  
    } else {  
        return cons(first(day).temp, cold(rest(day)));  
    }  
};
```

```
let precipitatingTemps: List<number> = tempify(precipitatingDays);
```

# Reducing to find all occurrences of snow on March 21<sup>st</sup>

Of all mornings of March 21sts in the past 10 years, how many times did it snow?



## Reducing to find all occurrences of snow on March 21<sup>st</sup>

```
let frequencyOfSnow = (temp: List<number>): number => {  
  let snowProbability: number = 0;  
  if (temp === null) {  
    return 0;  
  } else if (first(temp) <= 32) {  
    snowProbability = 1;  
  } else {  
    snowProbability = 0;  
  }  
  return snowProbability + frequencyOfSnow(rest(temp));  
};
```

```
let snowDayNum: number = frequencyOfSnow(precipitatingTemps);
```

# Conclusion

Today has been the only March 21<sup>st</sup> in the past 10 years that has started with snow!



# Function Literals

- Function literals are when we write functions **without** assigning them a name

```
let predFunction = (n: number): boolean => {  
    return n > 5;  
};
```

# Function Literals

- Function literals are when we write functions **without** assigning them a name

```
let predFunction = (n: number): boolean => {  
    return n > 5;  
};
```

# Function literals

- Before

```
let predFunction = (n: number): boolean => {  
    return n > 5;  
};  
let ourList = listify(6, 3, 7, 8, 2, 1, 9);  
filter(ourList, predFunction);
```

# Function literals

- After

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);
filter(ourList, (n: number): boolean => {
    return n > 5;
}));
```

# Function literals

- Let's simplify this even more

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);  
filter(ourList, (n: number): boolean => { return n > 5 });
```

# Type inference

- Declaring variables

```
let x: number = 0;
```

```
let s: string = "words go here";
```

```
let youKnowHowThisWorks: boolean = true;
```

# Type inference

- What types would these be?

```
let woo: _____ = "woo";
```

```
let spooky: _____ = 42;
```

```
let mystery: _____ = false;
```

# Type inference

- Typing is not a required element of programming in Typescript.
- However, sometimes it's nice to be verbose.

```
let x = 5;
```

```
let myWords = "cat dog mouse";
```

```
let surprise = listify("wow", "amazing", "cool");
```



# Type inference and Function literals

- Remember this?

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);
filter(ourList, (n: number): boolean => {
    return n > 5;
}));
```

# Type inference and Function literals

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);
filter(ourList, (n: number): boolean => {
    return n > 5;
}));
```

# Type inference and Function literals

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);  
filter(ourList, (n) => {  
    return n > 5;  
}));
```

# Type inference and Function literals

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);  
filter(ourList, (n) => { return n > 5; }));
```

# Type inference and Function literals

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);  
filter(ourList, (n) => { return n > 5; }));
```

# Type inference and Function literals

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);  
filter(ourList, (n) => n > 5);
```

# Motivation

Before

```
let ourList = listify(6, 3, 7, 8, 2, 1, 9);  
let predFunction = (n: number): boolean => {  
    return n > 5;  
};  
filter(ourList, predFunction));
```

After

```
filter(ourList, (n) => n > 5));
```

# Filtering with Function Literals

- Keeping only numbers that are greater than 5

```
let numbers = listify(1, 2, 3, 4, 5, 6, 7, 8, -500);  
filter(numbers, (n) => n > 5);
```

- Keeping only strings with length less than 5

```
let numbers = listify("", "hi", "heyoo", ... , "str");  
filter(numbers, (s) => s.length < 5);
```



# Mapping with Function Literals

- Turn a list of numbers into their string representation

```
filter(nums, (n) => "" + n);
```

- Turn a list of games into a list of points

```
filter(games, (g) => g.points);
```

# Reducing with Function Literals

- Find out if all of the booleans in a list are true

```
reduce(bools, (memo, b) => memo === true && b === true, true)
```

- Find the longest string in a list

```
reduce(strs, (memo, s) => {  
  if (memo.length <= s.length) {  
    return s;  
  } else {  
    return memo;  
  }  
}, "");
```

# Hands on

- Call the filter function with a list called “nums”, and a second argument that is a function literal.
- You should create the function literal, and filter out all of the numbers that are odd, keeping only even numbers in this list
- You should be able to do this without knowing what is in the nums list

# Hands on 2

- Call the map function with a list called “strs” and transform this list of strings into only the first character of the string.
- Remember: substr is a string method that takes in two arguments, the start index and the length of the substring.
- Ex:
- `“Hello”.substr(0, 2) === “He”`
- `let s: string = “Test”`
- `s.substr(1, 2) === “es”`