UMass Amherst | Manning College of Information & Computer Sciences

Lab 2: Type Signatures and More Higher-order Functions

Wednesday September 10, 2025

UMassAmherst | Manning College of Information & Computer Sciences

Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
    - Please sit with your group from last week.
- Discussion leaders will walk around and answer questions
- Solutions to exercises will be reviewed as a class
- Attendance taken at the end

UMassAmherst | Manning College of Information & Computer Sciences

Reminders

- Homework 2 is due Sunday (9/14) at 11:59pm
  - Come to office hours for help!
- If you need to miss lab and have a valid reason according to the syllabus (medical, other personal) please fill out the questionnaire on Canvas before the start time of your lab.
  - Waking up late, bus was late are NOT valid reasons to miss lab.
- Submit what you have at the end of lab to Gradescope. If you miss many submissions to Gradescope, we may penalize your lab grade.
  - Lab submission deadline is now 11:59pm; please submit by then!

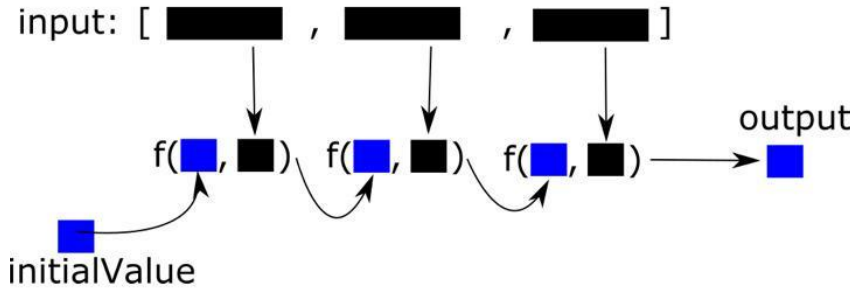
Today's Goals

- Practice with more higher-order functions
- Practice writing correct type signatures

UMassAmherst | Manning College of Information & Computer Sciences

Review of Reduce

```
function reduce<T, U>(
     a: T[],
     f: (acc: U, e: T) => U,
     init: U
): U {
     let result = init;
     for (let i = 0; i < a.length; ++i) {
          result = f(result, a[i]);
     }
     return result;
}
```



Reduce is used to combine array elements with the same function.

Example: Find the product of all elements of an array a = [3, 2, 6, 2, 2, 0]

```
a.reduce((prod, e) => prod * e, 1);
```

Exercise 1: Map, Filter & Reduce

Write a function <u>without using loops or recursion</u> to calculate the sum of the square roots of all positive numbers in an array.

Note: Use Math.sqrt() to calculate the square root

Example: Input: [-6, 0, 4, 16, -5] => Output: 2 + 4 = 6

Solution 1: Map, Filter & Reduce

Write a function without using loops or recursion to calculate the sum of the square roots of all positive numbers in an array.

Note: Use Math.sqrt() to calculate the square root.

Example: Input: [-6, 0, 4, 16, -5] => Output: 2 + 4 = 6

```
function sumSquaresPositive(nums: number[]): number {
    return nums
        .filter(num => num > 0)
        .map(num => Math.sqrt(num))  // Can we simplify this?
        .reduce((sum, num) => sum + num, 0);
}
```

Followup: Can you use just a single reduce on nums?

Alternative Solution

Followup: Can you use just a single reduce on nums?

Example: Input: [-6, 0, 4, 16, -5] => Output: 2 + 4 = 6

```
const getNum = (n: number) => n > 0 ? Math.sqrt(num) : 0;
function sumSquaresPositive(nums: number[]): number {
     return nums
          .reduce((sum, num) => sum + getNum(num), 0);
}
```

Exercise 2: Reduce

Write a function that counts how many pixels in an array are “mainly blue”.

- Input: An array of type Color[]   (where Color is a triple [R, G, B])
- Output: The number of pixels satisfying the following condition: the blue value is at least twice the red value and at least twice the green value
- Note: Define “`type Color = [number, number, number];`”

Can you solve the same problem for a 2D array of type Color[][] ?

UMassAmherst | Manning College of Information & Computer Sciences

Solution 2: Reduce

```
type Color = [number, number, number];
const isBlue = (c: Color) => c[2] >= 2*c[0] && c[2] >= 2*c[1] ? 1 : 0;

function countBlue(r: Color[]): number {
   return r.reduce((acc, c) => acc + isBlue(c), 0);
}
function countBlue2D(m: Color[][]): number {
   return m.reduce((acc, r) => acc + countBlue(r), 0);
}
```

UMassAmherst | Manning College of Information & Computer Sciences

Review of Type Signatures

We can infer types based on the operations done on values

```
// f(x: number, y: number): number     or  f: (number, number) => number
function f(x, y) {
  return x + (2*y);
  // product with y: y is number, x and result is number
}
```

Sometimes, we have several possibilities

```
// g: (number, number) => number    or    g: (string, string) => string
function g(x, y) { return x + y; }  // + can be string concatenation

// h(a: string): boolean   or   h<T>(a: T[]): boolean
function h(a) { return a.length > 5; } // both strings & arrays have length
```

The array could have any element type. We call T a **type variable.**
This lets us write **generic functions.**

Exercise 3: Type Signatures

What is the type signature for the following code?

```
const h = (a, g, f) => f(a.map(g)).filter(g);
```

Start by considering an arbitrary element type for the array, and continue to derive types for the map and filter callback functions.

Remember:

```
map<A,B>(arr: A[], f: (x: A) => B): B[]
filter<T>(arr: T[], f: (x: T) => boolean): T[]
```

UMassAmherst | Manning College of Information & Computer Sciences

Solution 3: Type Signatures

What is the type signature for the following code?

```
const h = (a, g, f) => f(a.map(g)).filter(g);
```

`a` must be an array of some type `T`.
Then `g: T => R` (as callback for map), resulting in an array of type `R[]`.
Function `f` takes the array of type `R[]` and returns an array (since we apply filter to the result). Since this array is filtered by `g: T => R`, its element type must be `T`, and `R = boolean`. The result is also an array of type `T[]`.

```
h: (a: T[], g: (x: T) => boolean, f: (a: boolean[]) => T[]): T[]
```

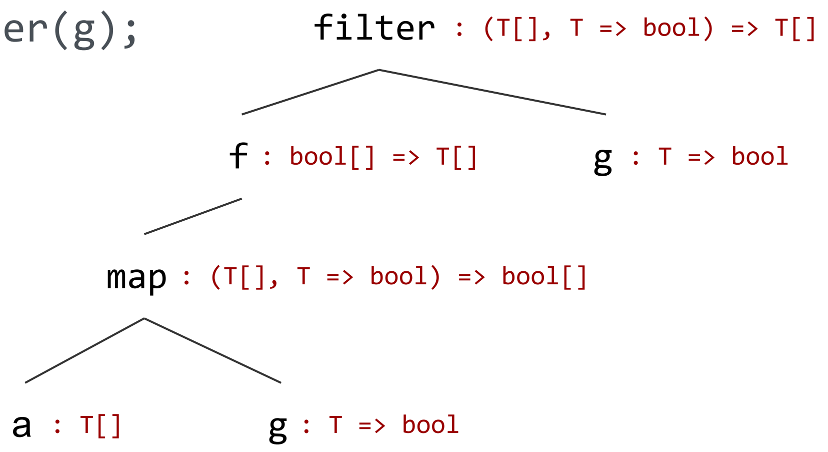
Solution 3: Type Signatures

UMassAmherst | Manning College of Information & Computer Sciences

```
const h = (a, g, f) => f(a.map(g)).filter(g);
```

Another method:

- Draw a tree for the call graph. Each node will be a function or an argument. Parents depend on children.
- Narrow down the types for each node, starting from either root or leaves
- Repeat until no more change is made.


filter : (T[], T => bool) => T[]
f : bool[] => T[]
g : T => bool
map : (T[], T => bool) => bool[]
a : T[]
g : T => bool