



Manning College of Information
& Computer Sciences

Programming Methodology
**Lab 2: Type Signatures and More
Higher-order Functions**

Wednesday February 11, 2025



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 will be taken at the end.

Reminders

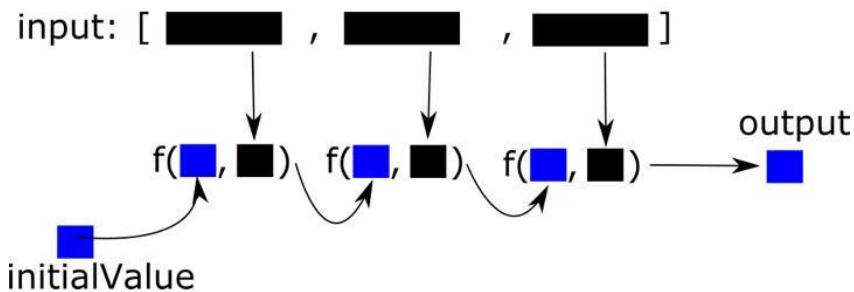
- Homework 2 is due Tonight (2/11) at 11:59pm.
 - Come to office hours for help!
- If you need to miss lab and have a valid reason according to the syllabus (medical, other personal) please fill out the lab excusal form on Canvas before the start time of your lab.
 - Waking up late or the bus being late 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.

Today's Goals

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

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: Reduce

Write a function **countBlue** that takes in an array **arr: Color[]** and returns the number of pixels that are “mainly blue”.

A color is “mainly blue” if the blue value is at least twice the red value and at least twice the green value. Recall that the color format is **[R, G, B]**.

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

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 2: More Reduce

Write a function **atLeastHalf** that takes in an array **arr** and a function **func**, which can be applied to an element of **arr** to produce a boolean.

The function should return **true** if **func** returns **true** for at least half of the array elements and **false** otherwise.

You will need to complete type signature for this function.

Use **reduce**.

Bonus: how could you implement this function without **reduce**?