



UMassAmherst

Manning College of Information  
& Computer Sciences

# Lab 1: HOFs and Type Signatures

Wednesday, September 3rd, 2025

# Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
  - Discussion leaders will assign groups today
  - Groups will be remade every third lab
- Discussion leaders will walk around and answer questions
- Solutions to exercises will be reviewed as a class
- Attendance taken at the end

# Reminders

- Please set up your development environment as soon as possible
- Homework 1 is due this Sunday (2/4) at 11:59pm
- 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 Groups

- You will now be assigned into your lab groups
- Please sit with your group each week
  
- Take the next 5 minutes to talk to each other
- Introduce yourselves!
  - Name, pronouns, major
  - Favorite household appliance
  - What was one fun thing you did over break?

# Today's Goals

- Set up coding environment
- Practice both higher-order functions and some TypeScript
- Walk out with some working code

# Writing and Running TypeScript

- Download the starter code from GitHub (linked on Canvas).
- Unzip the folder and open it in VSCode.
- Run **"npm install"** in the same directory as the package.json folder.
- When you are ready to submit, run **"npm run build:submission"**
- Upload the resulting zip file to the corresponding assignment on gradescope.
- Your lab leaders will walk you through this process for the first lab!

# Review of map

```
// Sample Implementation
// `.map` is a method on Arrays
function map<T, U>(
  a: T[],
  f: (x: T) => U
): U[] {
  const result: U[] = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(f(a[i]));
  }
  return result;
}
```

```
function double(x: number): number {
  return 2 * x;
}
```

```
const array = [1,2,3,4,5];
const newArray = array.map(double);
```

What is `newArray` ?

Reason about the code before typing and running it.

# Review of filter

```
// Sample Implementation
// `.filter` is a method on Arrays
function filter<T>(
  a: T[],
  f: (x: T) => boolean
): T[] {
  const result: T[] = [];
  for (let i = 0; i < a.length; ++i) {
    const x = a[i];
    if (f(x)) {
      result.push(x);
    }
  }
  return result;
}
```

```
function isEven(x: number): boolean {
  return x % 2 === 0;
}
const array = [1,2,3,4,5];
const newArray = array.filter(isEven);
```

What is `newArray` ?

Reason about the code before typing and running it.



# Programming Exercise 1

Write a function that takes an array of number arrays, and returns an array of number arrays where all negative values have been removed. Again, don't use any loops or recursion.

```
[[1,-2,3], [0], [0,1,2,3], []]
```



```
[[1,3],[0], [0,1,2,3], []]
```

A number array is typed as `number [ ]`, an array of those is `number [ ] [ ]`

# Programming Exercise 1

```
function keepNonNegativeValues(a: number[]): number[] {  
    return a.filter(x => x >= 0);  
}
```

```
function nonNegatives2D(arr: number[][]): number[][] {  
    return arr.map(keepNonNegativeValues);  
}
```

Important item here is to use the code developed in exercise 1. This is a good example of decomposing a problem into smaller problems, and of testing the smaller solutions before continuing.

# 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: Type Signatures

(a) What are the type signatures of `f` and `g`?

```
const a = [1,2,3,4];  
const b = a.filter(f);  
const c = b.map(g);
```

(b) Initialize variables `a`, `f`, and `g`, such that the code would run without error. The input and output types of `f` should be different from each other.

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

Remember:

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

## Solution 2: Type Signatures

(a) What are the type signatures of `f` and `g`?

```
const a = [1,2,3,4];  
const b = a.filter(f);  
const c = b.map(g);
```

`a` is an array of numbers.

Then `f: number => boolean` (as callback for `filter`), resulting in `b`, an array of numbers.

Now `g: number => T` (as callback for `map`), resulting in `c`, an array of type `T[]`.

## Solution 2: Type Signatures

(b) Initialize variables `a`, `f`, and `g`, such that the code would run without error. The input and output types of `f` should be different from each other.

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

One possible solution is:

```
const a = ["foo", "bar", "baz", "qux"];  
const f = (s: string): number => s[0].charCodeAt(0);  
const g = (n: number): boolean => n < 110;  
// h(a, f, g) == [102, 98, 98]
```

# Final Thoughts

- Think carefully about your approach before starting to code
- Start small and test often
- Try to make use of as much of your previous work as possible
- A lot of material all at once, come to office hours if you are confused
  - We would love to see you there! 😊
- It is okay if some of us take longer or don't finish before lab ends, keep working at it
  - Try not to get discouraged
- Starting the homework today is not required, but try to get into the habit of reading the instructions as soon as they are released