



UMassAmherst

Manning College of Information  
& Computer Sciences

Programming Methodology

## Lab 4: Closures and Mental Models

Wednesday September 24, 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 taken at the end

# Reminders

- Midterm 1 is next Tuesday (9/30) at 7-9pm
  - Exams from previous semesters can be found under Modules on Canvas
- Homework 4 will have 2 parts, with separate deadlines, both after the midterm.
  - Homework 4a has been released (due October 5th)
  - 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.

# Today's Goals

- Closures
- Mental Models

# Exercise 1: Closures

- Write a function that takes as argument an array of Boolean functions, all with the same argument type  $T$  and returns a Boolean closure with an argument  $x$  of type  $T$ . The closure should return true if and only if more than half of the functions in the array return true for  $x$ .
- Use reduce for implementation.

```
function mostTrue<T>(
  funarr: ((arg: T) => boolean)[]
): (arg: T) => boolean {
  // TODO
}
```

## Exercise 1: Solution

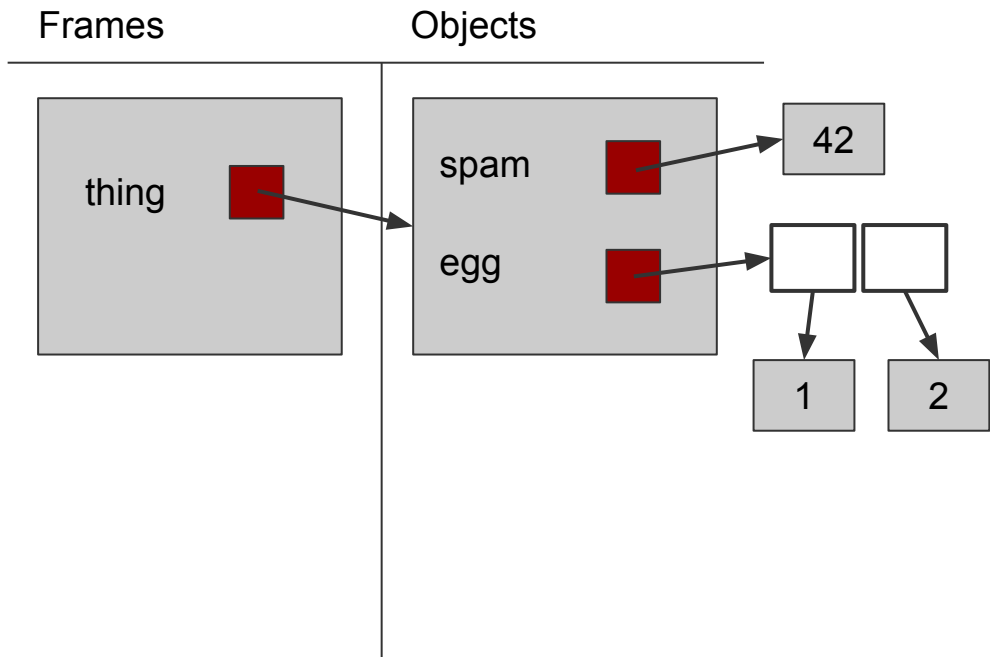
- Write a function that takes as argument an array of Boolean functions, all with the same argument type  $T$  and returns a Boolean closure with an argument  $x$  of type  $T$ . The closure should return true if and only if more than half of the functions in the array return true for  $x$ .
- Use reduce for implementation.

```
function mostTrue<T>(
  funarr: ((arg: T) => boolean)[]
): (arg: T) => boolean {
  return (x: T) => funarr.reduce((acc, f) => f(x) ? acc+1 : acc-1, 0) > 0;
}
```

## Exercise 2: Mental Models

As you may have seen towards the end of Lecture 7, the memory diagram for the following code is depicted on the right.

```
1 let thing = {  
2   spam: 42,  
3   egg: [1, 2]  
4 };
```

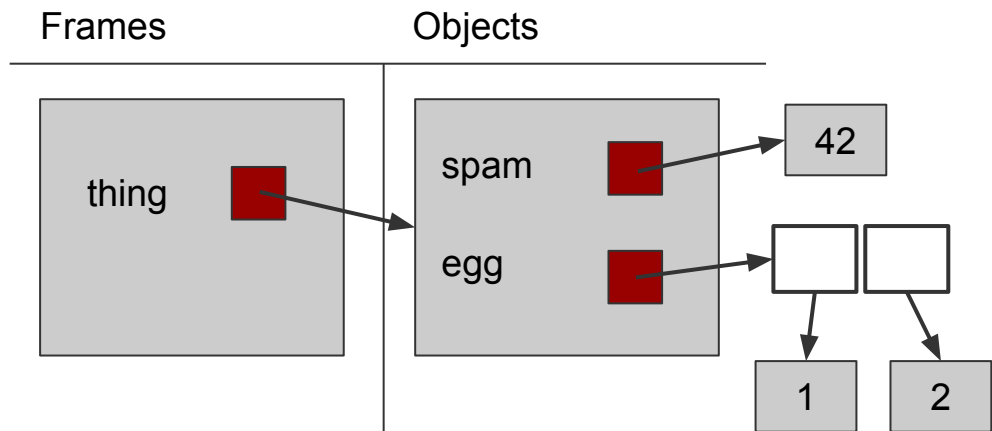


## Exercise 2: Mental Models

```
1 let thing = {  
2   spam: 42,  
3   egg: [1, 2]  
4 };
```

Draw the updated memory diagram after running lines 5-10 below:

```
5 let tmp = thing.egg;  
6 thing.egg[0] += 3;  
7 thing.egg = thing.spam;  
8 thing.spam = tmp;  
9 thing.spam.push(thing.egg);  
10 thing.spam.filter(x => x < 5);
```



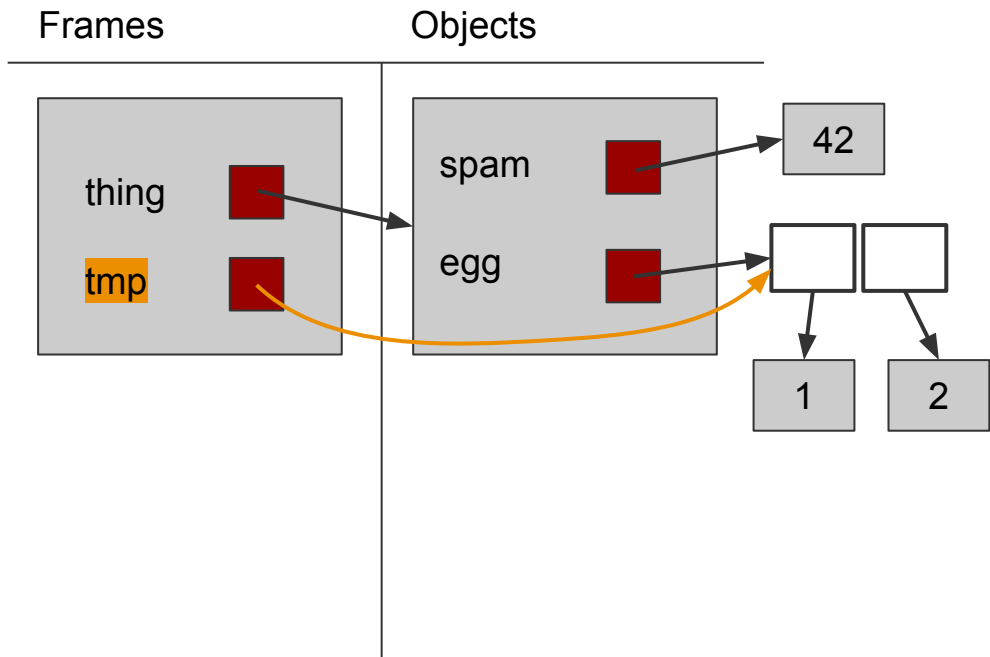
(lines 5-8 from lecture 7 exercise)



## Exercise 2: Solution

Line 5: variable tmp is initialized and the value stored in thing.egg (reference to array [1,2]) is stored in tmp.

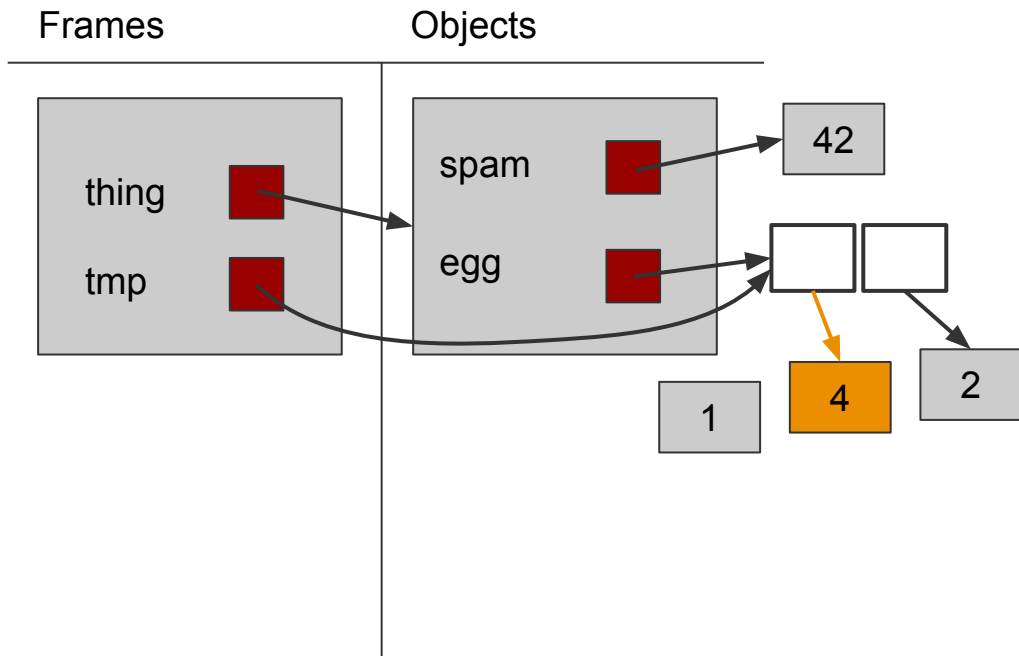
```
5 let tmp = thing.egg;  
6 thing.egg[0] += 3;  
7 thing.egg = thing.spam;  
8 thing.spam = tmp;  
9 thing.spam.push(thing.egg);  
10 thing.spam.filter(x => x < 5);
```



## Exercise 2: Solution

Line 6: the element at index 0 in the array referenced by `thing.egg` is incremented by 3.

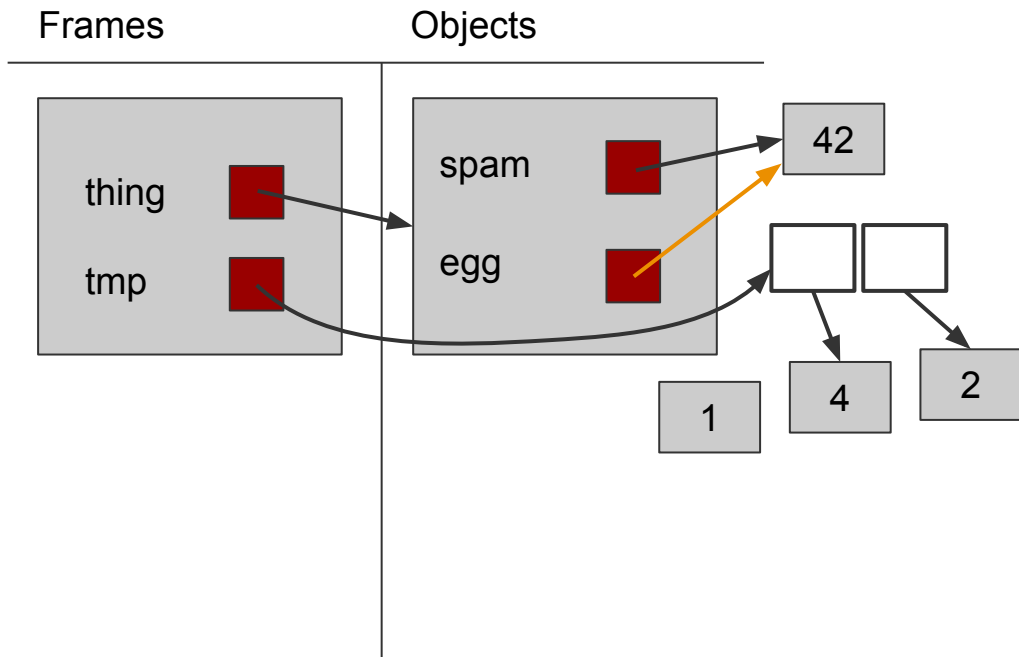
```
5 let tmp = thing.egg;  
6 thing.egg[0] += 3;  
7 thing.egg = thing.spam;  
8 thing.spam = tmp;  
9 thing.spam.push(thing.egg);  
10 thing.spam.filter(x => x < 5);
```



## Exercise 2: Solution

Line 7: the egg property of object thing is set to the value stored in thing.spam, which is the number 42.

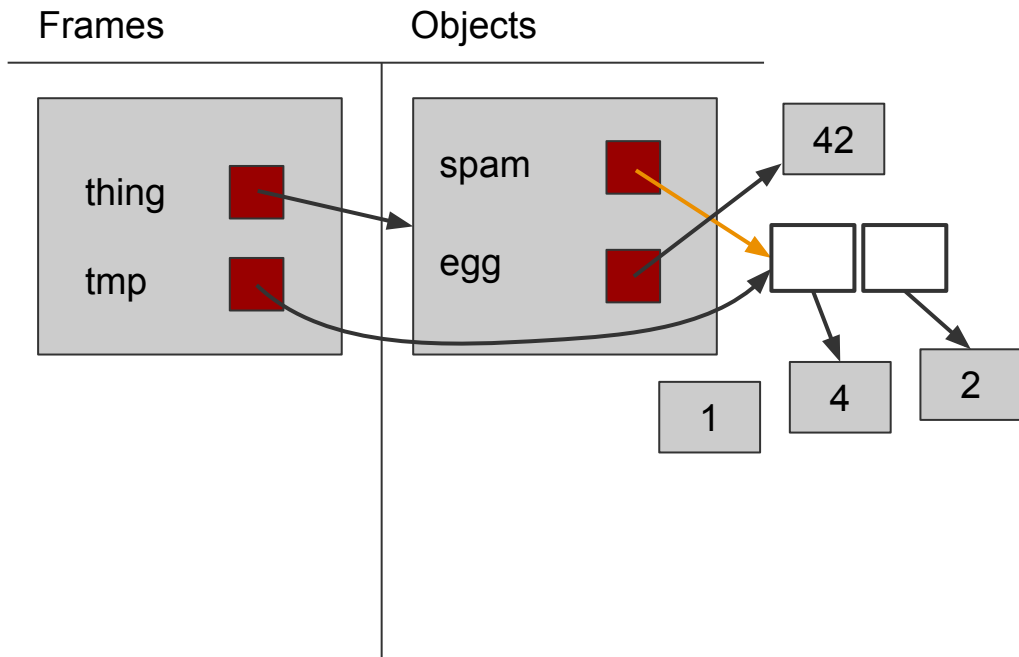
```
5 let tmp = thing.egg;  
6 thing.egg[0] += 3;  
7 thing.egg = thing.spam;  
8 thing.spam = tmp;  
9 thing.spam.push(thing.egg);  
10 thing.spam.filter(x => x < 5);
```



## Exercise 2: Solution

Line 8: the spam property of object thing is set to the value stored in tmp, which is the reference to the array [4, 2].

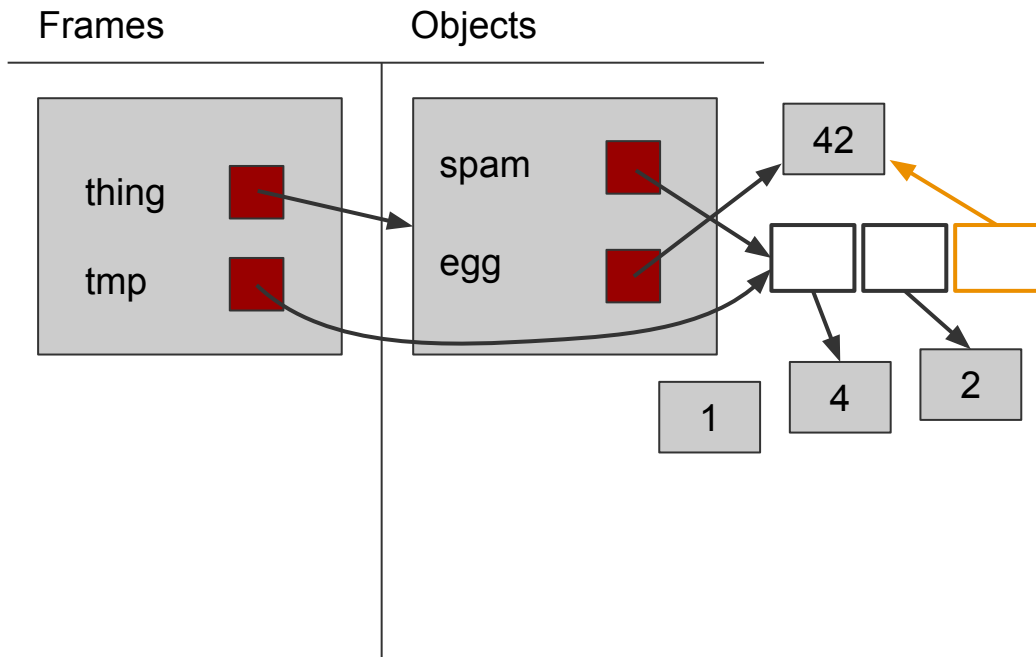
```
5 let tmp = thing.egg;  
6 thing.egg[0] += 3;  
7 thing.egg = thing.spam;  
8 thing.spam = tmp;  
9 thing.spam.push(thing.egg);  
10 thing.spam.filter(x => x < 5);
```



## Exercise 2: Solution

Line 9: the value stored in `thing.egg` (42) is added as an element to the array referenced by `thing.spam`.

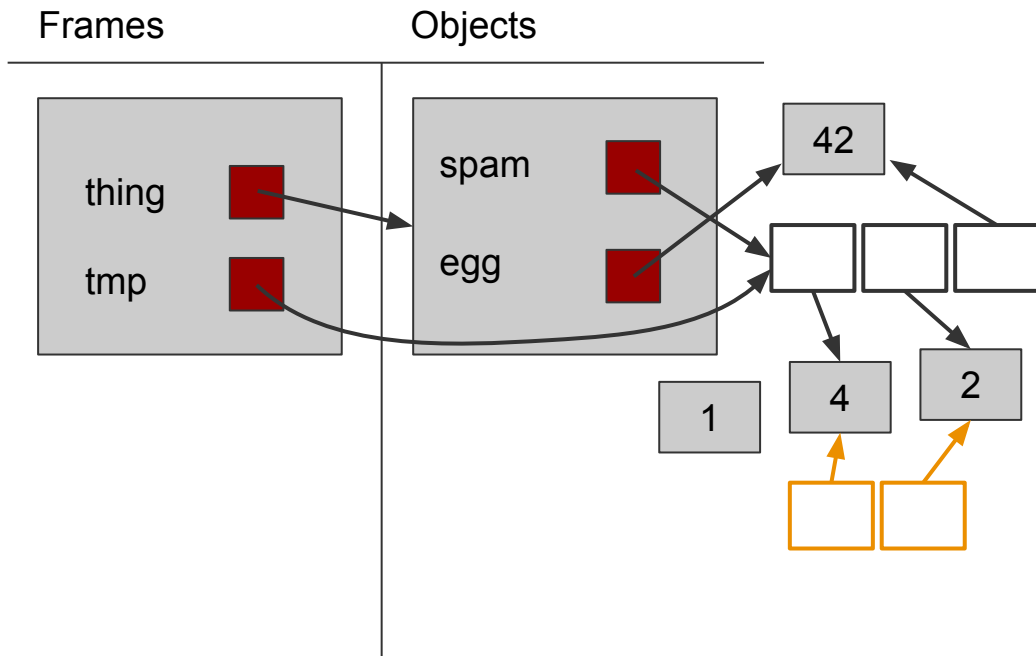
```
5 let tmp = thing.egg;
6 thing.egg[0] += 3;
7 thing.egg = thing.spam;
8 thing.spam = tmp;
9 thing.spam.push(thing.egg);
10 thing.spam.filter(x => x < 5);
```



## Exercise 2: Solution

Line 10: filter is called on the array referenced by thing.spam ([4, 2, 42]). A new array is created, with only the elements less than 5 ([4, 2]). The reference to this array is never stored.

```
5 let tmp = thing.egg;  
6 thing.egg[0] += 3;  
7 thing.egg = thing.spam;  
8 thing.spam = tmp;  
9 thing.spam.push(thing.egg);  
10 thing.spam.filter(x => x < 5);
```



## Exercise 3: More closures

- Write a function with no arguments that returns a closure with no arguments. When called the  $n$ th time ( $n \geq 1$ ), the closure should return the  $n$ th approximation for the number  $e$ :  $1 + 1/1! + 1/2! + \dots + 1/n!$
- Avoid needless recomputation in the factorial and in the sum.
- Example outputs: 2, 2.5, 2.666..., 2.70833..., 2.7166..., 2.718055..., etc.

## Exercise 3: Solution

- Write a function with no arguments that returns a closure with no arguments. When called the  $n$ th time ( $n \geq 1$ ), the closure should return the  $n$ th approximation for the number  $e$ :  $1 + 1/1! + 1/2! + \dots + 1/n!$
- Avoid needless recomputation in the factorial and in the sum.
- Example outputs: 2, 2.5, 2.666..., 2.70833..., 2.7166..., 2.718055..., etc.

```
function approxE(): () => number {  
  let n = 1, factorial = 1, res = 1;  
  // can you shorten this even further?  
  return () => {  
    factorial *= n++; // why n++ and not ++n?  
    return (res += 1/factorial);  
  }  
}
```