Programming Methodology

Lab 3: Closures and Lists

Wednesday February 18, 2026

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 3 is due tonight (2/18) 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.

Today's Goals

- Practice using closures
- Practice using lists

Review of Closures

A closure is a **function** bundled with its **surrounding environment**.

Closures can be used to **hide state** and dictate **specialized interaction**.

In JS/TS,

- A closure is created every time a function is created.
- Functions are **first-class**, so they can be used like any other value.
    - Unnamed functions are called **anonymous** functions.

Exercise 1 - Closures

Write a function **`approxE`** that takes no arguments.

It should return a closure that, when called the n^{th} time (n >= 1), returns the n^{th} approximation for the number e: 1 + 1/1! + 1/2! + ... + 1/n!

Avoid unnecessary recomputation in the factorial and in the sum.

Example outputs: 2, 2.5, 2.666..., 2.70833..., 2.7166..., 2.718055..., etc.

Review of Lists

Lists are **recursive**.

Lists are either

- empty
- or an **element** followed by a **list**

Lists are an **abstract data type** with the following methods:

- `list.isEmpty()`
  - returns `True` if the list is `empty()`
  - returns `False` if the list is `node(data,next)` [element followed by a list]
- `list.head()`
- `list.tail()`

Exercise 2 - Lists

Given two ordered lists, merge them such that the resulting list is an ordered list (ascending).

```
// merges two ordered lists

function merge(l1: List<number>, l2: List<number>): List<number>
```