## CS 360

**Winter 2018**

Programming Language Concepts

**CS 360-001** Tuesday/Thursday 15:30-16:50 (UCross 151)

**CS 360-002** Tuesday/Thursday 14:00-15:20 (UCross 151)

**CS 360-003** Tuesday 18:30-21:20 (UCross 153)

Instructor: Geoffrey Mainland
mainland@drexel.edu (mailto:mainland@drexel.edu)
Office: University Crossings 106
Office hours: Mondays 4pm–7pm; Thursdays 5pm–6pm.

Teaching Assistant: Xiao Han
CLC office hours: Wednesday 2pm–4pm; Thursday 12pm–2pm

# Homework 1: Scheme

Due Monday, January 14, 11:59:59PM EST.

Accept this assignment on GitHub Classroom here (https://classroom.github.com/a/pGEDFCux). The standard homework instructions (..) apply.

This assignment provides an introduction to list processing, functional programming, and Scheme. Implement, document (i.e., write a specification for), and test the following functions in scheme. Scheme includes some of the constructs below, but you **may not** use these implementations. Instead, you must implement them yourself. Make sure all functions are thoroughly tested.

You must implement the functions as specified. You may write other helper functions and define test data in your file, but you **may not** change the functions' names or the number or order of arguments. You must also write a short description of what the function does in a comment, as we did for the examples in class. If we ask you to implement a recursive solution and your solution is not recursive, you will receive no credit.

Also note:

- No error-checking is required; you may assume input is valid.
- You may **not** use Racket's built-in `log`, `exp`, or `expt` functions. Please write a recursive function instead of digging through the documentation trying to find a library function that does your job for you!
- You **may not** use `set!` or any form of mutable state in your solutions.
- We expect you to use proper Scheme style. We use proper Scheme style in class. You may also refer to this Scheme style guide (https://web.archive.org/web/20181115201554/http://community.schemewiki.org/?scheme-style) for general advice, but to be clear, it is *not* the case that anything goes ;)

You should complete the homework by modifying the file `hw1.rkt` that we provide as part of the assignment.

This assignment is worth 50 points. There are 50+1 possible points.

## Notes on style

The code we see in class adheres to standard Scheme style. In particular, the following are considered **bad** style:

1. Placing parentheses on lines by themselves.
2. Using `cond` when there are only two cases (use `if`)

You may lose one or more points for *each* problem exhibiting poor style.

If you have questions about style, please ask in office hours or on Piazza!

# Problem 1: `(sigma m n)` (10 points)

Define a **recursive** function `sigma` that takes two arguments, $m$ and $n$, and returns $\sum_{i=m}^{n} i$.

# Problem 2: `(log m n)` (10 points total)

Define a **recursive** function `log` that takes two arguments, $m$ and $n$, and returns the least integer $l$ such that $m^{l+1} > n$.

# Problem 3: `(choose n k)` (10 points total)

Define a **recursive** function `choose` that takes two arguments, $n$ and $k$, and calculates $\binom{n}{k}$, the binomial coefficient, which is also the number of ways of selecting $k$ items from a collection of $n$ items. Your `choose` should call itself recursively. An implementation that uses the closed-form formula for `choose` involving factorial will receive zero credit.

If you have forgotten the definition of the binomial coefficient, you may want to read the Wikipedia article (https://en.wikipedia.org/wiki/Binomial_coefficient). The following recurrence relation will be useful:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Make sure you handle the base cases:

$$\binom{n}{0} = \binom{n}{n} = 1$$

# Problem 4: `(binary n)` (10 points)

Define a function `binary` that takes a single argument $n$ and returns the number whose decimal representation is the binary representation $n$. For example, `(binary 2)` should evaluate to `10`.

Your function should be recursive. Your solution should not need to use strings or Racket's binary number type. Your function must return a **decimal** number.

# Problem 5: `(scan f z l)` (10 points)

The `scan` function scan takes a binary function `f`, a value `z`, and a list `l`, and returns the list $z, f(z, x_1), f(f(z, x_1), x_2), \ldots, f(f(\ldots f(f(z, x_1), x_2) \ldots, x_{n-1}), x_n)$ where $x_1, x_2, \ldots, x_n$ are the elements of the list `l`.

Examples:

```
(scan + 0 null) => (0)
(scan + 0 '(1 2 3 4 5 6)) => (0 1 3 6 10 15 21)
(scan * 1 '(1 2 3 4 5 6)) => (1 1 2 6 24 120 720)
```

This may remind you of the `reduce` function. You can think of `scan` as a version of `reduce` that returns a list of all the intermediate results of the reduction.

# Problem 6: Homework Statistics (1 point)

How long did spend on each problem? Please tell us in your `README.md`. You may enter time using any format described here (https://github.com/wroberts/pytimeparse).