Announcements

- **Midterm 1** is on **Monday, September 30 at 7:00 PM**. Look to “odyssey.uwaterloo.ca” for seating arrangements.

- **Assignment 3** is due on **Tuesday, October 1, at 9:00 PM**.

- Drop-down to **CS 115 deadline** is on **Wednesday, October 23**.

- The times and location of office hours are posted on the “Office and Consulting Hours” pages of the course website. Please email us at “cs135@uwaterloo.ca” to set up an appointment outside of these hours.

- **Monday** and **Tuesday** office hours have been moved from **MC 4065** to **MC 2063**.
Goals of this tutorial:
By the end of this tutorial you should be able to...

- Understand **List Structures**
- **Create** and **Navigate** Lists
Review Clicker Question

What is the next step of this evaluation?

(or false false)

A (or false)

B (or)

C (false)
Review: Lists

A list is a nested structure, meaning it is made by nesting a value with another list. Think of Russian wood dolls:

- A list of 3 dolls, is a single doll with 2 dolls inside of it.
- A list of 2 dolls, is a single doll with 1 doll inside of it.
- A list of 1 doll, is a single doll with nothing inside of it.
- A list of 0 dolls is a special. It is an empty list, however, it is still a list.
Review: Making Lists

In beginner language you can make a list using the \((\text{cons} \ \text{value} \ \text{list})\) function. A list of integers would follow the rules below, and could look like the example below.

\[
\text{;; A (list of Int) is one of:}
\]
\[
\text{;; * empty}
\]
\[
\text{;; * (cons Int (listof Int))}
\]
\[
\text{define no-scope (cons 3 (cons 6 (cons 0 empty))))
\]
Review: Making lists of Different Types

You may also create lists that hold value of different types.

the rules for such a list are stated below, along with an example.

;;; A (list of Any) is one of:
;;; * empty
;;; * (cons Any (listof Any))

(define truth (cons "true" (cons 'true (cons true empty)))))
Problem 1: 3D → 2D

You have been provided a set of 3-dimensional vectors in the form: (cons x (cons y (cons z empty))).
Define a function that consumes a vector and produces a list of the X and Z values.

(change-dimension (cons 1 (cons .2 (cons −3 empty)))))
→ (cons 1 (cons −3 empty))
Problem 2: Perpendicular?

What story is more sad than two fate-bound individuals meeting once and never again?

Perpendicular lines are a pair of lines that cross each other at a right angle (90 degrees).

Define a function that takes in two pairs of points and determines if the lines formed are perpendicular to each other.
Problem 2: Perpendicular? - Help

Note: Two lines are perpendicular to each other if the multiplication of their slopes are equal to -1.

Example:

Slope1 = $x/2$

Slope2 = $-2x$

Slope1 * Slope2 = $1/2 * -2 = -1$

Thus, these two lines form a right angle.
Problem 2: Perpendicular? - Hint 1

So our first challenge is to find the slope between two points. Define a function that determines the slope formed by two points (Assume the slope exists).
Problem 2.1: Find Slope - Design Recipe

;;; (find-slope alpha omega) Consumes two points alpha and omega to find the slope formed by them.
;;; find-slope: (listof Num) (listof Num) → Num
;;; Requires: (length of alpha, omega = 2)
;;; Example:
(check-expect (find-slope (cons 1 (cons 2 empty)) (cons 2 (cons 4 empty))) 2)

(define (find-slope alpha omega) . . .)

;;; Tests:
(check-expect (find-slope (cons 1 (cons 1 empty)) (cons 2 (cons 1 empty))) x)
(check-expect (find-slope (cons 1 (cons 1 empty)) (cons 1 (cons 2 empty))) x)
(check-expect (find-slope (cons 1 (cons 1 empty)) (cons 1 (cons 1 empty))) x)
Problem 2: Perpendicular - Design Recipe

(define point1 (cons 2 (cons 4 empty)))
(define point2 (cons 0 (cons 0 empty)))
(define point3 (cons 5 (cons -10 empty)))
(define point4 (cons 0 (cons 3 empty)))
(define point5 (cons 4 (cons 0 empty)))
(define point6 (cons -1 (cons 0 empty)))

;; (perpendicular? p1 p2 p3 p4) Determine whether 2 lines formed
;; by 4 points p1, p2, p3, p4 are intersected with right angle.
;; perpendicular?: (listof Num) (listof Num) (listof Num) (listof Num) → Bool
;; requires: p1 p2 are not the same, p3 p4 are not the same
;; Example:
(check-expect (perpendicular? point1 point2 point3 point4) false)

(define (perpendicular? p1 p2 p3 p4) . . . )

;; Test:
(check-expect (perpendicular? point4 point2 point5 point6) true)
(check-expect (perpendicular? point5 point6 point2 point4) true)
Problem 3: Snowball

Snowballing is when one event happens, and then it is followed by another of the same event, only this time with greater magnitude because it includes the added effects of the previous effect. We can imitate this principal in racket with the help of lists.

Define a function that takes in a list with 3 values and produces a single value. With that value being the result of taking the first value as the base and the value directly after as its power.

When repeating the process with the result of the first calculation as the base and the third value as its power.

(Use a helper function in your approach, even if you think it is unnecessary)
Problem 3: Snowball

;; (snowball snow) Consumes a list of numbers snow and produces the snowball value.
;; snowball: (listof Num) → Num
;; Requires: (length of snow = 3)
;; Example:
(check-expect (snowball (cons 2 (cons 3 (cons 2 empty)))) 64)

(define (snowball base lst) ...)

;; Tests
(check-expect (snowball (cons 9000 (cons 9000 (cons 0 empty)))) 1)

...