Fall 2024 Extra Credit

- This is an **optional** homework assignment that—if completed—will provide up to 5% of the total course grade in extra credit.
- You will be asked to write a set of FuncLang functions and upload to Canvas a **single file** containing **all** necessary FuncLang code (all requested function definitions and any helpers your code uses).
- This file should be named **bst.scm**
- You are allowed to use any FuncLang code that has been presented in the textbook or in the lecture slides, but **you must include them in your submission** (we will not be tracking those down to include ourselves).
- You are **not** allowed to use any FuncLang code drawn from any other sources (no StackOverflow, ChatGPT, etc.)
- You are expected to complete this homework on your own, more like an exam question than a typical homework. As such, the TAs and instructors will provide less guidance on this than they would on a typical homework assignment.

- For this assignment, we ask you to implement a **binary search tree (BST)** in FuncLang. A BST is a special form of a binary tree that stores its nodes in a sorted order, allowing for fast lookup, insertion, and removal operations. In a typical binary search tree implementation:
  - Each node is assigned a **value**.
  - Each node has at most one **left** child and at most one **right** child.
  - The value of a given node is **greater than or equal to** the value of any node in the subtree rooted in its **left** child.
  - The value of a given node is **less than** the value of any node in the subtree rooted in its right child.
  - An in-order traversal of the tree (evaluating children left-to-right) will provide the list of values in the tree sorted in ascending order.

- Your FuncLang implementation of a BST must provide the following interface:
  - A constructor function named `bst` which accepts a single list of numbers as an argument and returns a BST containing the numbers in that list, added **in order from left to right**. (NOTE: the list may be empty). A BST where the root node has no value is "empty".
  - An `insert` function which accepts two arguments—(1) a BST and (2) a list of numbers to insert—and returns a new BST which is the result of inserting each of the numbers from the passed list **in order from left to right** into the passed BST (the list may be empty).
  - A `getlist` function which accepts a single BST as an argument and returns the list of values stored in the passed BST in ascending order. If the BST is empty, this function will return an empty list.
  - A `gettree` function which accepts a single BST as an argument and returns a nested list structure which shows the tree structure of the BST. For a given BST, the returned list will contain exactly three values, in this order:
    * The value of the root node of the BST.
    * The list representing the tree structure for the BST rooted at the left child
    * The list representing the tree structure for the BST rooted at the right child

    If the given BST is empty (i.e., the root node does not have a value), then this function will return an empty list.
- Example behavior:

```
$ run bst.scm

$ (define t0 (bst (list)))

$ (define t1 (insert t0 (list 5)))

$ (gettree t1)
(5.0 () ())
$ (define t2 (insert t1 (list 3 9 6 5)))

$ (getlist t2)
(3.0 5.0 5.0 6.0 9.0)
$ (define t3 (insert t2 (list 7 4)))

$ (getlist t3)
(3.0 4.0 5.0 5.0 6.0 7.0 9.0)
$ (gettree t3)
(5.0 (3.0 () (5.0 (4.0 () ()) ())) (9.0 (6.0 () (7.0 () ())) ()))
```