**CIS 61 - Lab 09 - Generators - Linked Lists - Trees**

### 

### Q1: Scale

Implement the generator function scale(s, k), which yields elements of the given iterable s, scaled by k. As an extra challenge, try writing this function using a yield from statement!

|  |
| --- |
| **def** **scale**(s, k):  """Yield elements of the iterable s scaled by a number k.  >>> s = scale([1, 5, 2], 5)  >>> type(s)  <class 'generator'>  >>> list(s)  [5, 25, 10]  >>> m = scale(naturals(), 2)  >>> [next(m) for \_ in range(5)]  [2, 4, 6, 8, 10]  """  "\*\*\* YOUR CODE HERE \*\*\*" |

### Q2: Link to List

Write a function link\_to\_list that takes in a linked list and returns the sequence as a Python list. You may assume that the input list is shallow; none of the elements is another linked list.

Try to find both an iterative and recursive solution for this problem!

|  |
| --- |
| **def** **link\_to\_list**(link):  """Takes a linked list and returns a Python list with the same elements.  >>> link = Link(1, Link(2, Link(3, Link(4))))  >>> link\_to\_list(link)  [1, 2, 3, 4]  >>> link\_to\_list(Link.empty)  []  """  "\*\*\* YOUR CODE HERE \*\*\*" |

### Q3: Cumulative Sum

Write a function cumulative\_sum that mutates the Tree t so that each node's label becomes the sum of all labels in the subtree rooted at the node.

|  |
| --- |
| **def** **cumulative\_sum**(t):  """Mutates t so that each node's label becomes the sum of   all labels in the corresponding subtree rooted at t.  >>> t = Tree(1, [Tree(3, [Tree(5)]), Tree(7)])  >>> cumulative\_sum(t)  >>> t  Tree(16, [Tree(8, [Tree(5)]), Tree(7)])  """  "\*\*\* YOUR CODE HERE \*\*\*" |

### Q4: Is BST

Write a function is\_bst, which takes a Tree t and returns True if, and only if t is a valid binary search tree, which means that:

* Each node has at most two children (a leaf is automatically a valid binary search tree)
* The children are valid binary search trees
* For every node, the entries in that node's left child are less than or equal to the label of the node
* For every node, the entries in that node's right child are greater than the label of the node

Note that, if a node has only one child, that child could be considered either the left or right child. You should take this into consideration.

*Hint:* It may be helpful to write helper functions bst\_min and bst\_max that return the minimum and maximum, respectively, of a Tree if it is a valid binary search tree.

|  |
| --- |
| **def** **is\_bst**(t):  """Returns True if the Tree t has the structure of a valid BST.  >>> t1 = Tree(6, [Tree(2, [Tree(1), Tree(4)]), Tree(7, [Tree(7), Tree(8)])])  >>> is\_bst(t1)  True  >>> t2 = Tree(8, [Tree(2, [Tree(9), Tree(1)]), Tree(3, [Tree(6)]), Tree(5)])  >>> is\_bst(t2)  False  >>> t3 = Tree(6, [Tree(2, [Tree(4), Tree(1)]), Tree(7, [Tree(7), Tree(8)])])  >>> is\_bst(t3)  False  >>> t4 = Tree(1, [Tree(2, [Tree(3, [Tree(4)])])])  >>> is\_bst(t4)  True  >>> t5 = Tree(1, [Tree(0, [Tree(-1, [Tree(-2)])])])  >>> is\_bst(t5)  True  >>> t6 = Tree(1, [Tree(4, [Tree(2, [Tree(3)])])])  >>> is\_bst(t6)  True  >>> t7 = Tree(2, [Tree(1, [Tree(5)]), Tree(4)])  >>> is\_bst(t7)  False  """  "\*\*\* YOUR CODE HERE \*\*\*" |