term test #1 — Wednesday — cheat aid sheet 8½ x 11"
50 minutes, 4 questions, 5 marks each.

A2

Lab — yes!

CSC148 winter 2014

linked structures

week 7

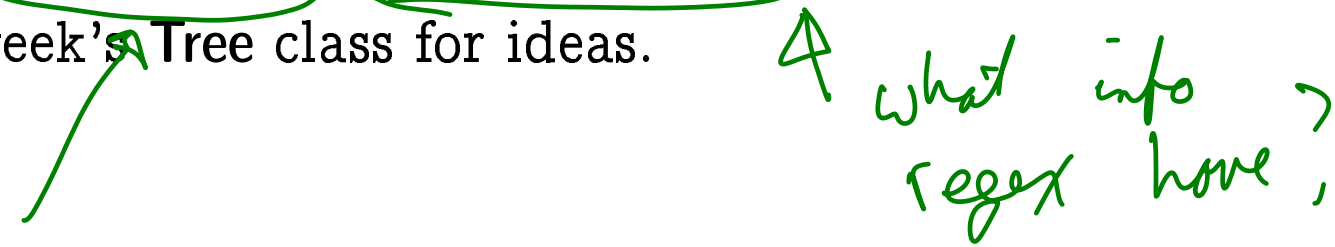
Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/W14/
416-978-5899

February 24, 2014

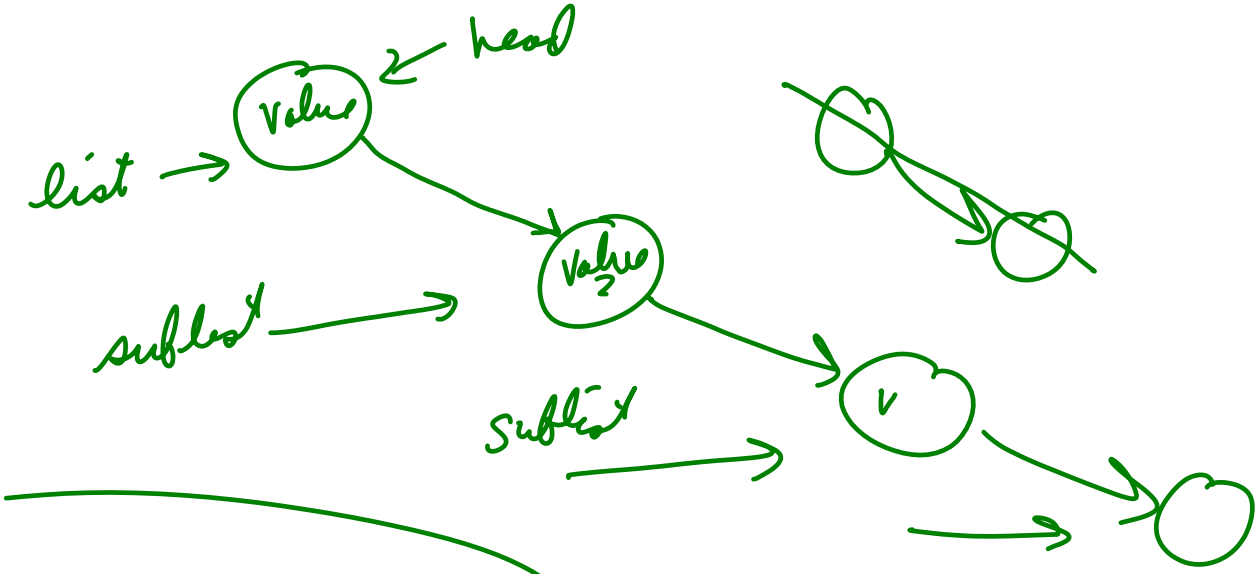
Outline

regular expressions

Start by designing a class hierarchy. What information is needed for each type of regular expression tree? What information is specialized? What's general? Look at last week's **Tree** class for ideas.



linear trees?

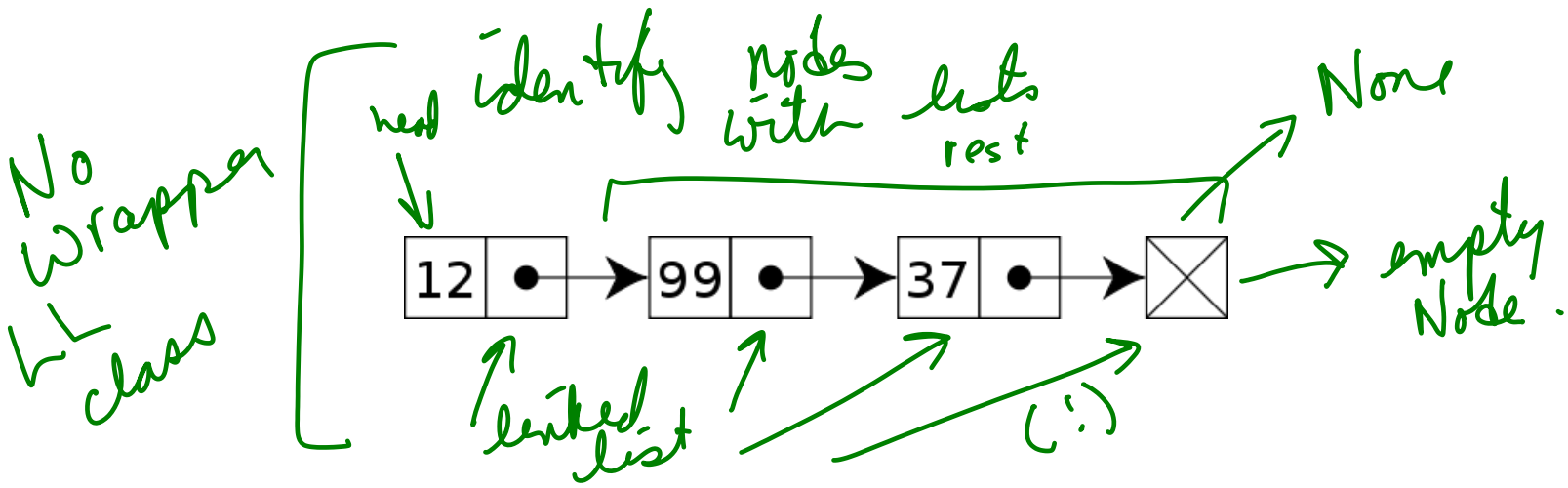


Trees of arity (branching factor) 1 can be thought of as a sequence of lists. Every node has no more than one child, and every node (other than the lone leaf) has no less than one child.

linked lists, conceptually

- **data:** Sequence of nodes, each with a **head** (value) and a reference to **rest** (its successors).

- **operations:** **prepend(object)**, **__contains__(value)**



LinkedList class

```
class LinkedList:
    """Linked list class"""

    def __init__(self: 'LinkedList', head: object=None,
                 rest: 'LinkedList'=None) -> None:
        """Create a new LinkedList.
        head - first element of list, absent in empty list
        rest - list of remaining elements, absent in empty list
        """
        # a list is empty if and only if it has no rest
        self.empty = rest is None
        if not self.empty:
            self.head, self.rest = head, rest
        elif head:
            raise Exception('Non-empty list requires a rest')
```

design choices

[5, None, "store"]

LinkedList(None, LinkedList())

None → X

LinkedList initialization reveals design choices

- **LinkedList()** creates an empty list — how do you know?
- empty lists are special — where can they occur, and what might they mean?
- it's possible for **head** to refer to None — why might you want this?
- **rest** refers to another **LinkedList** with the same structure

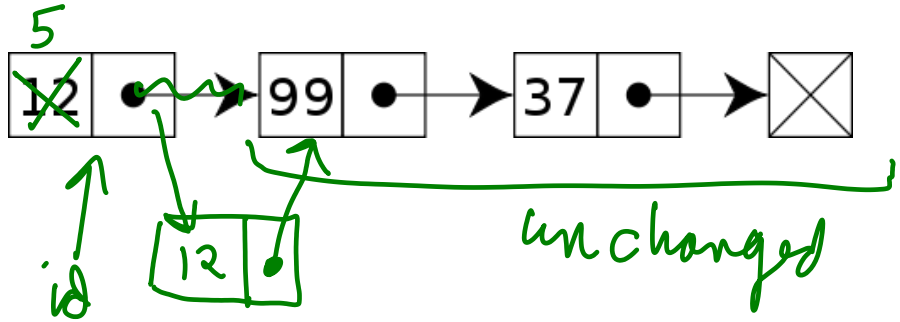
This isn't the only design for a linked list, for example How to think like a computer scientist show the "wrapper" approach.

implement prepend(head)

main goals are to preserve the list identity (same id) and preserve the previous contents

- start the rest of the list with the current attributes (shallow **copy** them)
- change the current head to the one passed in
- change the current rest to the copy!

Try drawing the result of **prepend(5)**



implement __contains__

def __contains__(self, value) → bool:

There are really three possibilities:

- this **LinkedList** is empty, so it can't possibly contain the value being sought
- the head of this **LinkedList** matches the value we seek
- or it's in the rest
- the head doesn't match, so check whether the rest contains the value we seek