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

Computer Science
UNIVERSITY OF TORONTO

Outline

regular expressions

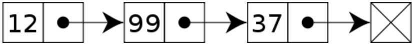
Start by designing a class hierarchy. What information is needed for each type of regular expression tree? What information is specialized? What's general?
Look at last week's **Tree** class for ideas.

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(value), __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 - The first element of the linked list,
        Not present if self will be the empty linked list.
        rest - The linked list that contains the elements after
        Not present if self will be the empty linked list."""
        self.empty = (head is None) and (rest is None)
        if not self.empty:
            self.head = head
            if rest is None:
                self.rest = LinkedList()
            else:
                self.rest = rest
```

Computer Science
UNIVERSITY OF TORONTO

design choices

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

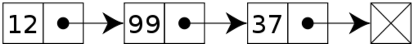
This isn't the only design for a linked list, for example How to think like a computer scientist show the "wrapper" approach.

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__

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
- the head doesn't match, so check whether the rest contains the value we seek