Python lists

• We learned about how Python lists are implemented.
  – Keep list elements in consecutive slots, starting at slot 0.
  – Allocate extra space to expand into.

• Inserting or deleting at the front of the list is thus slow.
  – Everything else must shift up or down.
  – Time grows linearly with size of list.

• But inserting or deleting at the back is faster.
  – Usually, no shifting is required.
A different kind of list

- If we don’t need the items to be in consecutive slots in memory, no shifting required.
- But we’ll have to keep track of where the items are!
- Solution: With each item store the id of the next one.
Linked List
A common pattern with Python lists

```python
i = 0
while i < len(lst):
    ... lst[i] ...
    i = i + 1
```
A common pattern with Python lists

• We have a variable that starts at the beginning of the list.

• On each iteration,
  – We use the variable to do something with the current item in the list.
  – We also advance it to the next item in the list.

• We stop when there are no more items to visit.
A common pattern with linked lists

curr = lst._first
while curr is not None:
    ... curr.item ... 
curr = curr.next
What did we give up?

• What do we gain by using a linked list rather than a Python list?
• What do we lose?

• Later, we’ll learn a data structure called a Tree that ties to have the gain without the loss.