# §12.10-12.11, Py ch17: Linked Lists

26 Nov 2007
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

# Review last time (12.1-12.5)

- Pointers (in Modula-2 and C)
  - Creating pointers, dereferencing pointers
  - Assignment compatibility
  - Pointer arithmetic
  - NIL (in C: NULL)
- Static vs. dynamic allocation of memory
  - Activation records
  - Stack, stack pointer
- Dynamic variables: NEW(), DISPOSE()

# Quiz09

- What are namespaces?
- Define and contrast: scope vs duration of a variable
- Say a class method references a variable name. According to Python's scope rules, what is the order of namespaces searched to resolve that name?
- Define and contrast: static vs. dynamic variables
- Name the area of memory used for dynamic variables
- What are pointers?

# What's on for today (12.8-12.12)

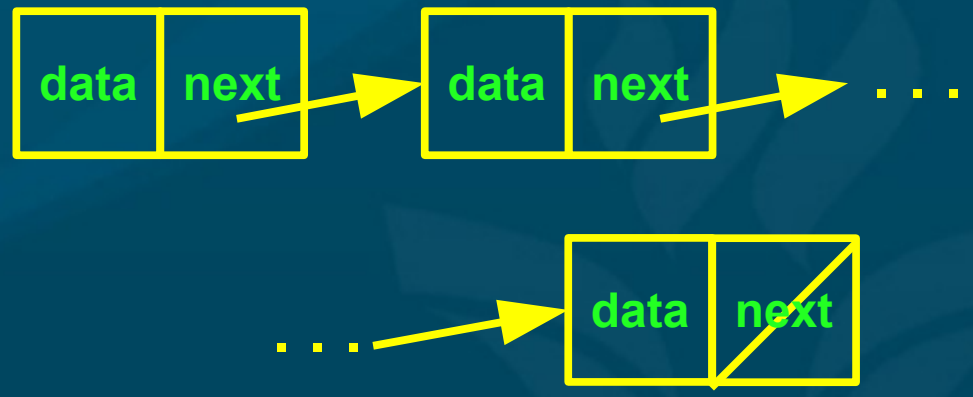
- Linked lists
  - Type definition, creating a new list
    - Inserting in nth position
    - Insert at head, append to tail
    - Deleting
  - Algorithmic efficiency
  - Circularly linked lists
  - Bidirectional lists

# Linked lists: creating

- A linked list is a dynamic ADT where each item in the list contains a pointer to the next item:

```
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

n1 = Node()
n2 = Node()
n1.next = n2
```



# Operations on linked lists

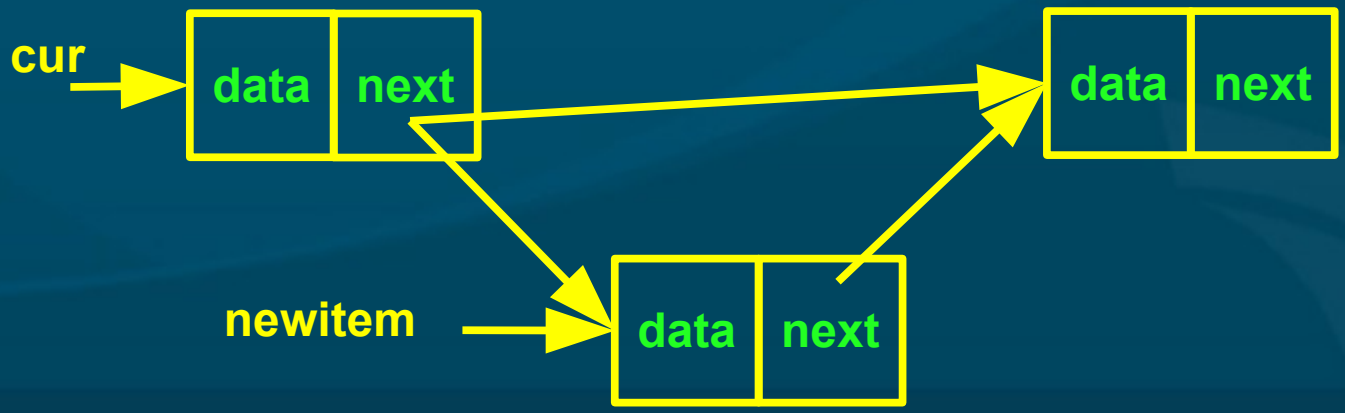
- Index into list (get a reference to $n^{th}$ node)
- Print out the list
- Search list for given data (cargo/payload)
- Insert a new node into a linked list
- Delete a node from a linked list
  - By index (0, 1, 2, ...) or by cargo

# Inserting a node into a linked list

- Follow pointers to get to the right spot
  - Create a new node with the given cargo
  - Thread new node into the list

```
newitem = Node(data)
newitem.next = cur.next
cur.next = newitem
```

  - What about inserting at head of list?



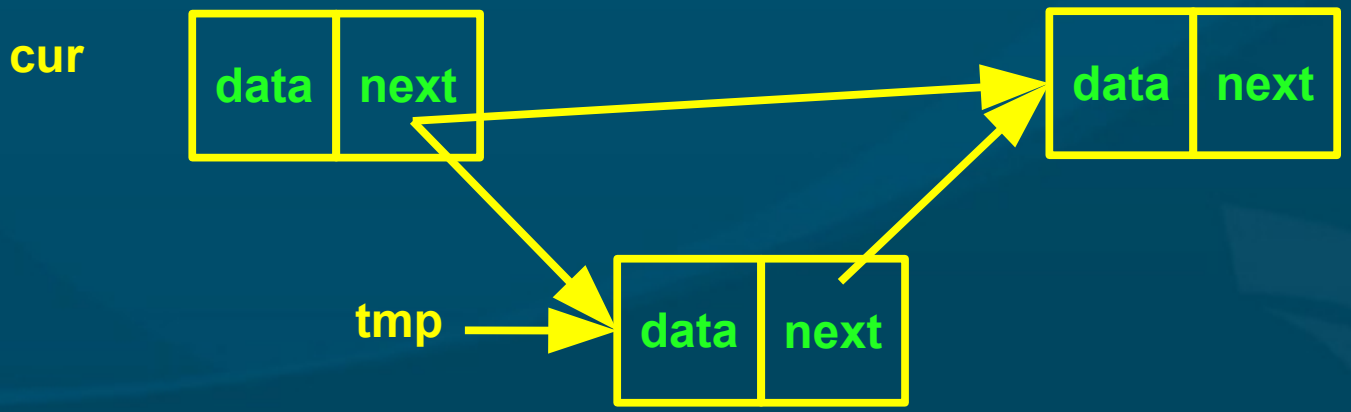
# Insert() method: code

```
def insert (self, n, data=None):
    """Insert a new node into linked list at position n."""
    newitem = Node(data)
    if n == 0:                          # new head: modify self
        newitem.next = self
        self = newitem
    else:
        cur = self
        for idx in range(n-1):          # get to proper position
            cur = cur.next
        newitem.next = cur.next
        cur.next = newitem
```

# Deleting from a linked list

- Follow pointers to find the item we want to delete
  - Sew up links to skip over the item
  - Deallocate the item from memory

```
tmp = cur.next
cur.next = tmp.next
del tmp
```



# Linked lists: algorithmic efficiency

- Big-O notation: O(n) means # operations varies linearly with n
- For a linked list with n items:
  - Insert at head: don't have to traverse list: O(1)
  - Append to tail: must walk list: O(n)
  - General insert:
    - Worst-case: O(n)
    - Average-case: O(n/2), which is also O(n)
  - Deleting: also O(n)
- Double-headed list (keep a tail pointer):
  - Speeds up append-to-tail to O(1)

# Variants of linked lists

- Circularly linked list:
  - tail.next = head
  - How to keep from infinite loop?
- Bidirectional linked list:

```
class Node:
    def __init__(self, data=None, prev=None, next=None):
        self.data = data
        self.prev = prev
        self.next = next
```

# TODO

- Lab09 due Wed:
  - Knight's tour
- HW10 due Fri
- Lab10 due next week:
  - Implement one of your old Lab04-07 in M2
  - Full lab-writeup (may reuse old writeup)
- Paper due next Mon!