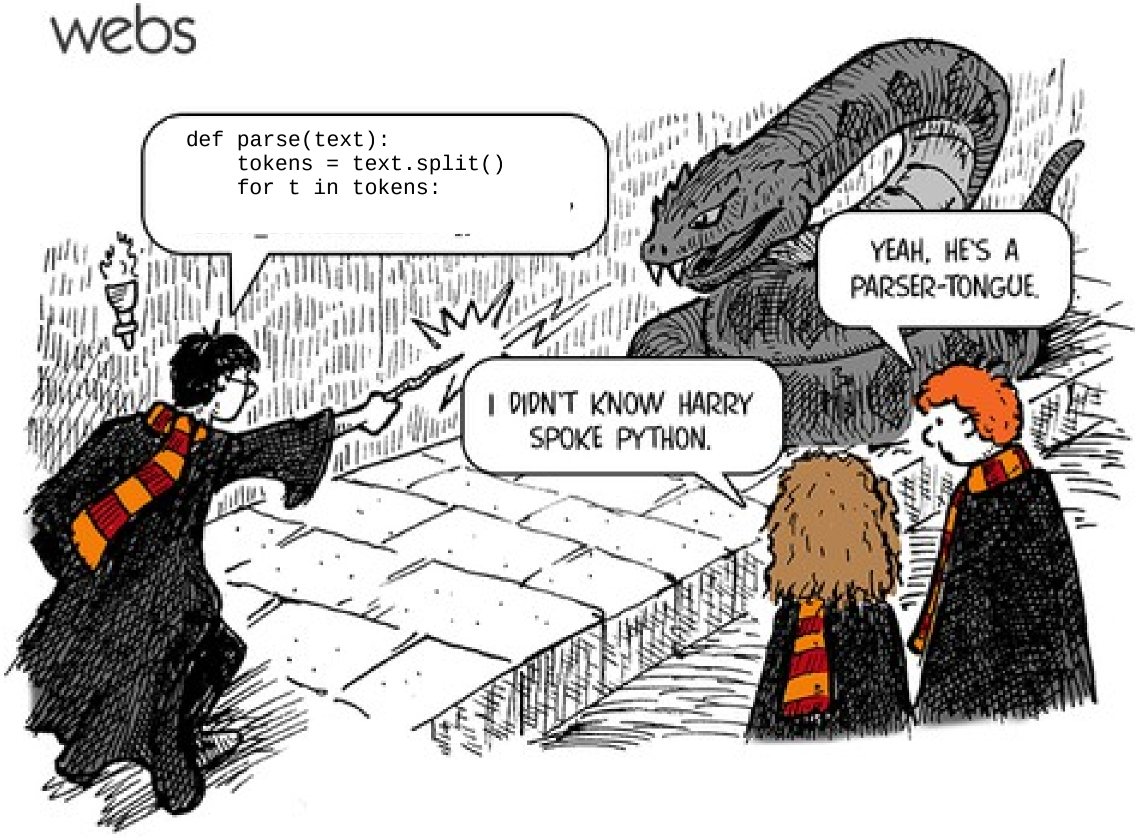
CS240
Fall 2014

Mike Lam, Professor



```
def parse(text):
    tokens = text.split()
    for t in tokens:
```

webs
YEAH, HE'S A PARSER-TONGUE.
I DIDN'T KNOW HARRY SPOKE PYTHON.

Linked Lists

Upcoming Career Fair

- CS Career Fair
  - Date: Wed, October 15, 10am-3pm
  - Location: nTelos Room (ISAT 259)
  - Looking for CS majors/minors only
  - Jobs and internships

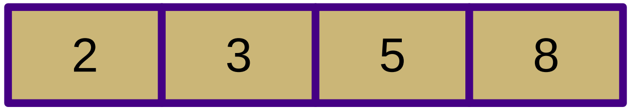
- Interview prep session
  - Date: Mon, October 6, 6:30pm
  - Location: HHS 2203
  - Free food!
  - Bring your resumé and cover letter!

Retrospective

- Arrays are great
  - O(1) access time to any element
  - Amortized O(1) insertion and removal
  - Referential arrays allow arbitrary-sized objects
- There are still disadvantages
  - Requires large chunks of reserved memory
  - Insertion/removal in the middle is expensive

Retrospective

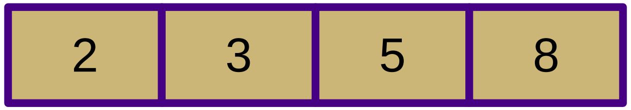
- Goal: Do less work when inserting and removing in the middle of our lists



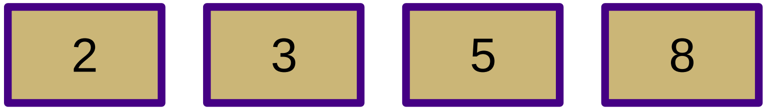
2	3	5	8

Retrospective

- Goal: Do less work when inserting and removing in the middle of our lists

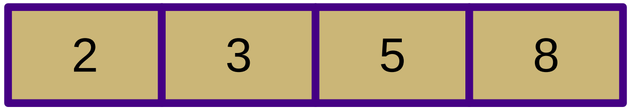


- Let's "pull apart" the array

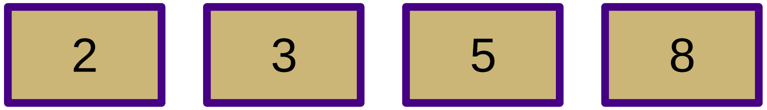


Retrospective

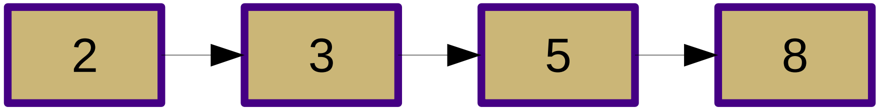
- Goal: Do less work when inserting and removing in the middle of our lists



- Let's "pull apart" the array

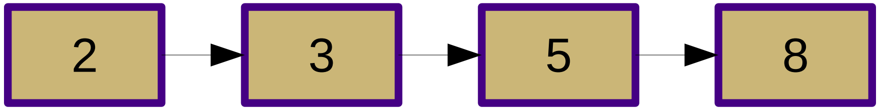


- And add links between all the items



Linked Lists

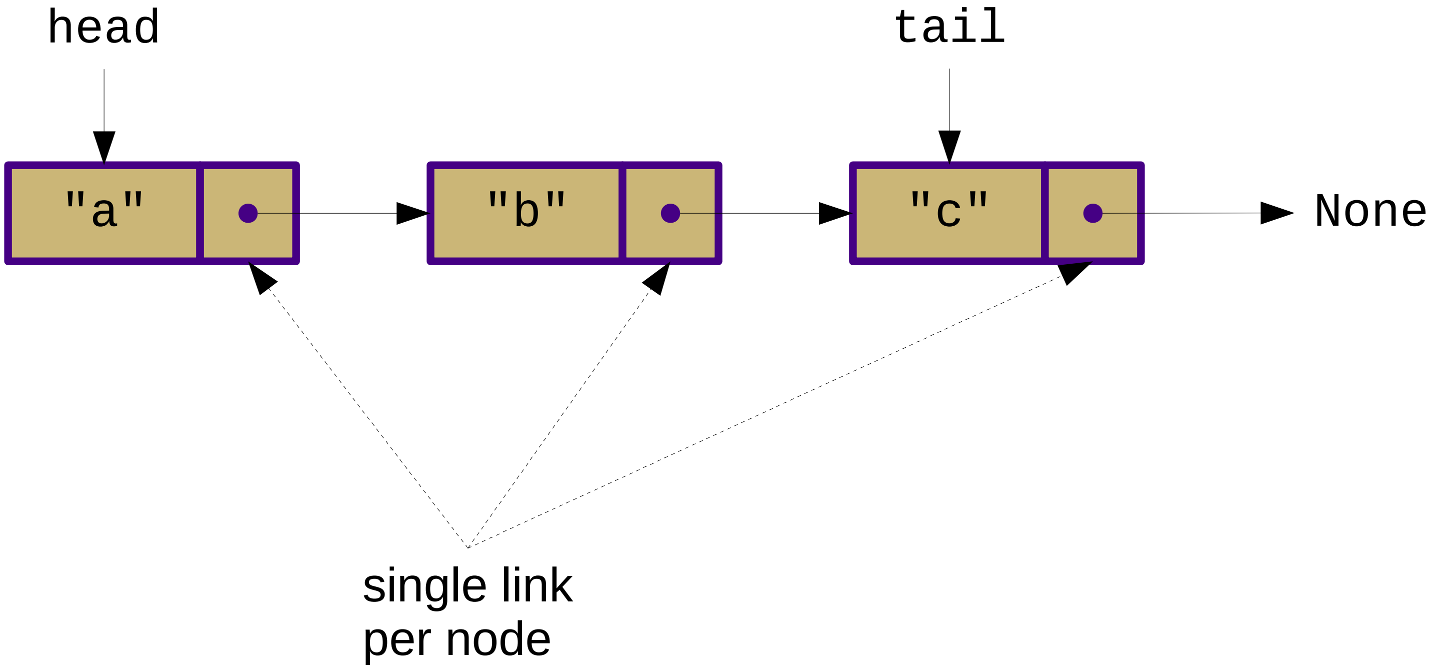
- This is a "linked list"



- Every item has a "next" pointer/reference
  - Last item has a null (`None`) "next" pointer
- Add and remove items by manipulating the pointers
- Keep external pointers to the beginning ("head") and end ("tail") of the list

Singly-Linked Lists

- Singly-linked list:



Singly-Linked Lists

- Inserting at the head:
  - `newest = Node(e)`
  - `newest.next = L.head`
  - `L.head = newest`
  - `L.size += 1`

Singly-Linked Lists

- Inserting at the tail:
  - `newest = Node(e)`
  - `newest.next = None`
  - `L.tail.next = newest`
  - `L.tail = newest`
  - `L.size += 1`

Singly-Linked Lists

- Removing from the head:
  - `if L.head is None:`
    - `raise Exception("List is empty")`
  - `L.head = L.head.next`
  - `L.size -= 1`

Singly-Linked Lists

- Removing from the tail:
  - `if L.tail is None:`
    - `raise Exception("List is empty")`
  - `L.tail = ???`
  - `L.size -= 1`
- Problem: Can't access previous node

Singly-Linked Lists

- Removing from the tail:
  - `if L.tail is None:`
    - `raise Exception("List is empty")`
  - `L.tail = ???`
  - `L.size -= 1`
- Problem: Can't access previous node
  - Solution: Track previous nodes as well
    - (doubly-linked lists)

Challenge

- Given a singly-linked list called "data", write a snippet of code that will print out all of the values in the list

Singly-Linked Lists

- Insert: O(1)
  - if you have a reference to the location
  - O(n) if the new location is index-based or the list needs to be sorted
- Delete: O(1)
  - if you have a reference to the item
  - O(n) if you have to look for it
- Indexed access or search: O(n)

Linked Stack

- Consider stack implementation using a singly-linked list

Linked Stack

- Consider stack implementation using a singly-linked list
  - Insert and remove at the head
  - Push, pop, and top are O(1)

Linked Queue

- Consider queue implementation using a singly linked list

Linked Queue

- Consider queue implementation using a singl
linked list
  - Insert at tail, remove from head
    - Can't remove from the tail!
  - Enqueue, dequeue, and first are O(1)

Looking ahead

- What if we kept two pointers?
  - "next" and "prev"
  - This is a "doubly-linked list"
- What if tail.next pointed to the head?
  - This is a "circularly-linked list"
- What if we kept multiple pointers to places further down the list?
  - This is a "skip list"

Reminder

- PA2 is due next Wednesday (Oct 8) at 23:59 (11:59pm)