The Rise of Google

Announcement

- HW 2 due today.
- HW 3 will be posted later this week, when we figure out CADE lab's Python, SciPy, NumPy versions
- Please go to TA's office hours
- If you have trouble locating the TA during the office hours, please email me
- Canvas might be down this morning...if you have trouble submitting your homework.

More on Lists

More on Lists

list.**append**(*x*): Add an item to the end of the list; equivalent to a[len(a):] = [x].

list.**extend**(*L*): Extend the list by appending all the items in the given list; equivalent to a[len(a):] = L.

list.**insert**(*i*, *x*): Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.**remove**(*x*): Remove the first item from the list whose value is *x*. It is an error if there is no such item.

list.**pop**([*i*]): Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.**index**(*x*): Return the index in the list of the first item whose value is *x*. It is an error if there is no such item.

list.**count**(*x*): Return the number of times *x* appears in the list.

More on Lists

list.**sort**(*cmp=None*, *key=None*, *reverse=False*)

Sort the items of the list in place (the arguments can be used for sort customization, see **sorted()** for their explanation).

list.**reverse**()

Reverse the elements of the list, in place.

https://docs.python.org/2/tutorial/datastructures.html

```
course_list = ['biology', 'math', 'computer science']

print course_list
print course_list[0]

course_list.append('history')
print 'course_list after appending', course_list

todo_list =['laundry', 'clean kitchen']
course_list.extend(todo_list)
print 'course_list after extending to todo_list:'
print course_list
```

```
number_list = [20, 18, 16, 12, 25]

print 'before:',number_list
number_list.insert(0, 7)
print 'after insertion:', number_list

number_list.insert(3, 8)
print 'after 2nd insertion:', number_list
```

```
number_list = [20, 18, 16, 12, 25]

print 'before:',number_list
number_list.insert(0, 7)
print 'after insertion:', number_list

number_list.insert(3, 8)
print 'after 2nd insertion:', number_list

number_list.remove(16)
print 'after 1st removal:', number_list
number_list.remove(17)
print 'after 2nd removal:', number_list
```

```
before: [20, 18, 16, 12, 25]
after insertion: [7, 20, 18, 16, 12, 25]
after 2nd insertion: [7, 20, 18, 8, 16, 12, 25]
after 1st removal: [7, 20, 18, 8, 12, 25]
ValueError: list.remove(x): x not in list
```

```
number_list = [20, 18, 16, 12, 25]

print 'before:',number_list
number_list.pop(0)
print 'after pop:',number_list

number_list.pop()
print 'after 2nd pop:',number_list

number_list.pop(2)
print 'after 3rd pop:',number_list
```

before: [20, 18, 16, 12, 25]
after pop: [18, 16, 12, 25]
after 2nd pop: [18, 16, 12]
after 3rd pop: [18, 16]

```
number_list = [20, 18, 16, 12, 25]

print 'list:',number_list
print 'The index of 16 in the list:',number_list.index(16)

number_list.extend([16,16])
print number_list
print number_list.count(16)
```

list: [20, 18, 16, 12, 25]
The index of 16 in the list: 2
[20, 18, 16, 12, 25, 16, 16]
3

```
number_list = [9, 10, 99, 2, 7, 6];

print number_list
number_list.sort()
print number_list
```

[9, 10, 99, 2, 7, 6]
[2, 6, 7, 9, 10, 99]

```
number_list = [9, 10, 99, 2, 7, 6, 2, 3];

print number_list
print number_list.index(2)
print number_list.index(99)
print number_list.index(8)
```

```
[9, 10, 99, 2, 7, 6, 2, 3]
3
2
ValueError: 8 is not in list
```

```
number_list = [9, 10, 99, 2, 7, 6, 2, 3];


print 8 in number_list
print 99 in number_list
```

False
True

Searching

Searching a List

- Search: seeing if a list contains a certain item
- Sequential search:
  - Start at the beginning of the list and compare each item in order, until the item is found
  - At most, requires as many steps as there are items in the list
- Can we do better?

Binary Search

- First, sort the list
  - Expensive operation, worth it if many searches will be done
- Can tell if a sub-group contains an item by looking at the first and the last items in the (sorted) sub-group

25, 5, 12, 13, 17, 20, 28

Can you tell if this group of numbers contain 42?

5, 12, 13, 17, 20, 25, 28

Can you tell if this group of numbers contain 42?

5, 12, 13, 17, 20, 25, 28

Easy: looking at the end values, 5 and 28. 42 is outside the range.

Binary Search Pseudo Code

```
While (the list is not empty)
    Set mid to the "middle" item in the list.
    If (mid is the item being searched for)
        End the search (return true).
    If (mid is less than the item)
        Set the list to be the second half of the list.
        (Throw away the first half of the list.)
    If (mid is greater than the item)
        Set the list to be the first half of the list.
        (Throw away the second half of the list.)

End the search (return false).
```

Example 1: binary search

Search for: hippo

Sorted list:

[ant cat chicken cow deer dog fish goat horse rat snake]

Iteration 1:

mid is dog, *updated list is* [fish goat horse rat snake]

Iteration 2:

mid is horse, *updated list is* [fish goat]

Iteration 3:

mid is goat, *updated list is* []

return false

Example 2: binary search

Search for: deer

Sorted list:

[ant cat chicken cow deer dog fish goat horse rat snake]

Iteration 1:

mid is dog, *updated list is* [ant cat chicken cow deer]

Iteration 2:

mid is chicken, *updated list is* [cow deer]

Iteration 3:

mid is deer, *return true*

Example 3: binary search

Search for number 8

Search for: 8

Sorted list:

[1, 4, 8, 12, 14, 15, 16, 24, 37, 39]

Iteration 1:

mid is 15, updated list is [1, 4, 8, 12, 14]

Iteration 2:

mid is 8, return true

Binary Search Speed

How quickly can binary search find an item? floor(logn+1)

What if you doubled the size of the list?

If you only wanted to search a list for one thing, would you use binary or sequential search?

Sorting

Review

- Searching lists
  - Find existence of an item in a list
    - Sequential search: might need to look at each item
    - Binary search:
      - Need a sorted list
      - Can rule out half the list each time
      - How do we sort the list?

Sorting

Arrange items in numerical order

- If the items are not numbered, need to compute a numerical score
- Example: the ranking of a webpage
- Example: sort people by their height
- Example: Alphabetical book titles

This is different from the other meaning of the work: to organize (e.g. sort your recycling)

Sorting Algorithms

- Different approaches have different characteristics
- They can be compared on
  - # of steps
  - Memory usage
  - Speed on random vs. almost ordered data

Selection Sort

Selection Sort

Rough idea: pick the smallest remaining and put it in the ordered group

Algorithm for Selection Sort

Input: a list, *Unsorted*, of unordered items
Initialization: set *Sorted* to empty

while (items remain in *Unsorted*)
    find the smallest item in *Unsorted*
    put that item on the end of *Sorted*

Output: the finished list *Sorted*

Selection Sort Example

Unsorted	Min	Sorted
5 2 1 4 3	1	1
5 2 4 3	2	1 2
5 4 3	3	1 2 3
5 4	4	1 2 3 4
5	5	1 2 3 4 5
-	-	1 2 3 4 5

More Details

Our algorithm glosses over some important details: Find smallest

Finding Smallest: sequential Scan

```
Set min to first item in list
Set index to 2      (the line above checks the 1st item)
While (index is not past the end of the list)
    If the item at index is less than min
        Set min to the item at index
    increment index
```

Finding Smallest Example

List	Index	Item at index	Min
4 5 3 1 2	1	4	4
	2	5	4
	3	3	3
	4	1	1
	5	2	1
		-	1

How many Steps to Sort?

How many steps to find the smallest in a list?

What is a step?

How many times do you have to search for the smallest?

How would this work on 10 items vs. 1,000,000 items?

Insertion Sort

Another Search: Insertion Sort

- Alphabetizing books or papers
- Take the top of one pile and insert it in the correct place in the sorted pile
- Start looking from the back of the sorted pile to find the correct place
- Insertion sort
  - Insert the next value in the correct spot

Unsorted	Top Value	Insert After	Sorted
5 2 1 4 3	5	front	5
2 1 4 3	2	front	2 5
1 4 3	1	front	1 2 5
4 3	4	2	1 2 4 5
3	3	2	1 2 3 4 5
- EXAMPLE		-	1 2 3 4 5

Computation

How many steps do selection and insertion sort require?

Are any cases better than others?

Sorting Demo

Look at the various approaches in the sorting applet:

http://www.sorting-algorithms.com

Merge Sort Game

Merge Sort Sketch

1. Divide the list into the smallest unit (1 element)
2. Compare each element with the adjacent list to sort and merge the two adjacent lists
3. All the elements are sorted and merged

https://commons.wikimedia.org/wiki/File:Merge-sort-example-300px.gif#/media/File:Merge-sort-example-300px.gif

Game Rule

1. Need 8 Volunteers to demonstrate merge sort
2. Bonus: 1 point each

THANKS!

Any questions?

You can find me at
beiwang@sci.utah.edu

http://www.sci.utah.edu/~beiwang/teaching/cs1060.html

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash