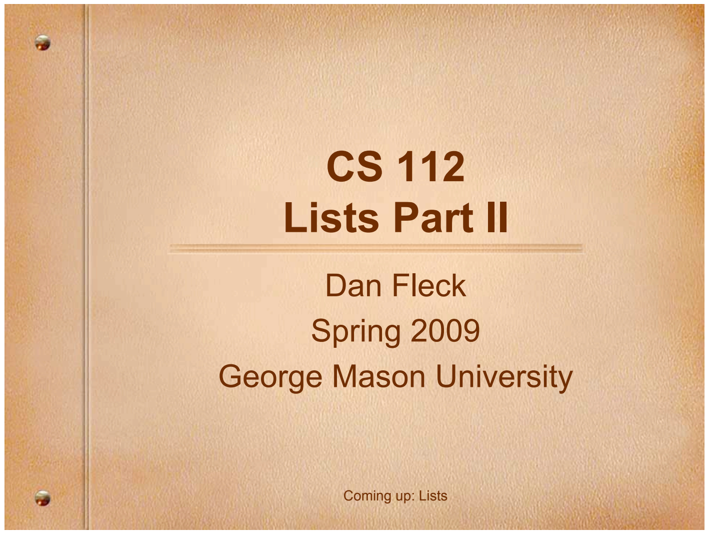


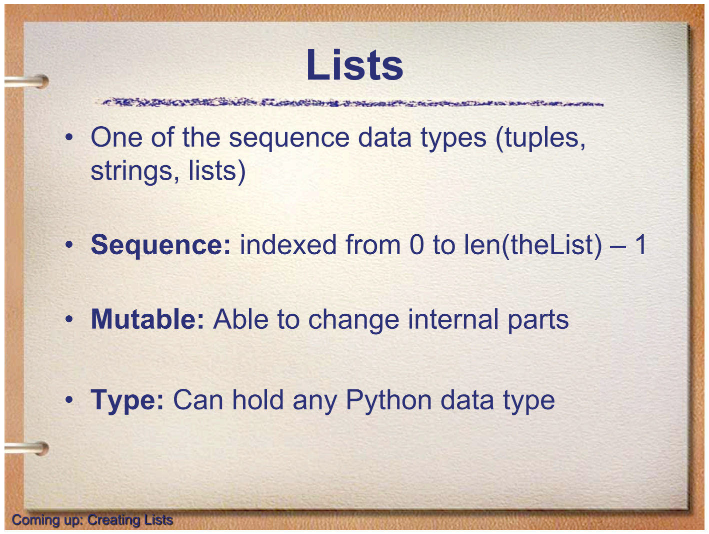
CS 112
Lists Part II

Dan Fleck

Spring 2009

George Mason University

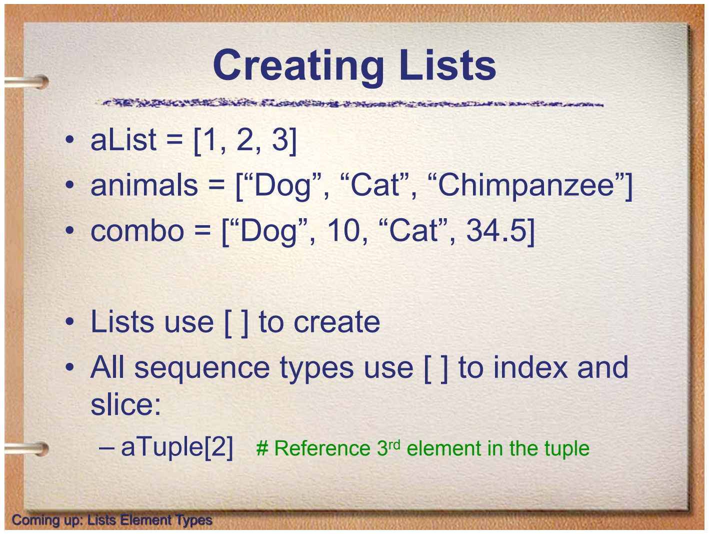
Coming up: Lists



Lists

- One of the sequence data types (tuples, strings, lists)
- **Sequence:** indexed from 0 to len(theList) – 1
- **Mutable:** Able to change internal parts
- **Type:** Can hold any Python data type

Coming up: Creating Lists

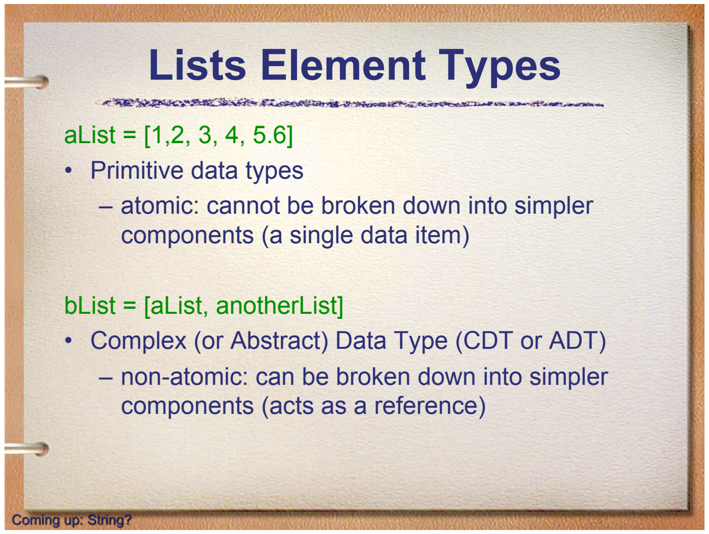


Creating Lists

- aList = [1, 2, 3]
- animals = [“Dog”, “Cat”, “Chimpanzee”]
- combo = [“Dog”, 10, “Cat”, 34.5]

- Lists use [ ] to create
- All sequence types use [ ] to index and slice:
  - aTuple[2]   # Reference 3rd element in the tuple

Coming up: Lists Element Types



Lists Element Types

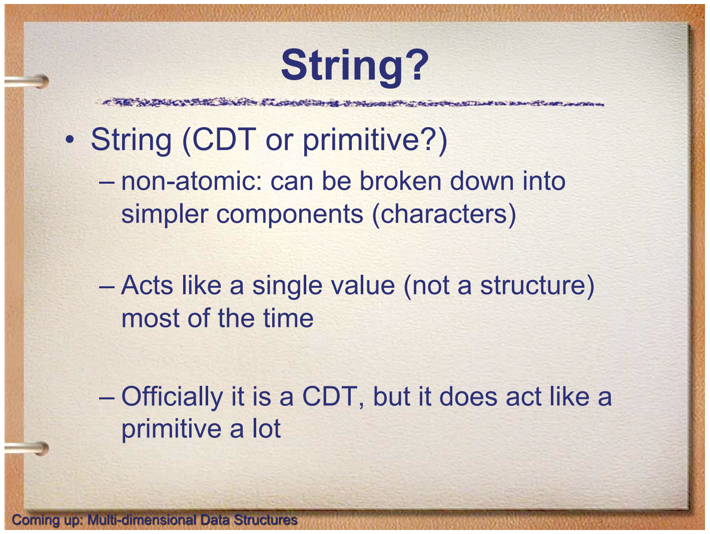
aList = [1,2, 3, 4, 5.6]

- Primitive data types
  - atomic: cannot be broken down into simpler components (a single data item)

bList = [aList, anotherList]

- Complex (or Abstract) Data Type (CDT or ADT)
  - non-atomic: can be broken down into simpler components (acts as a reference)

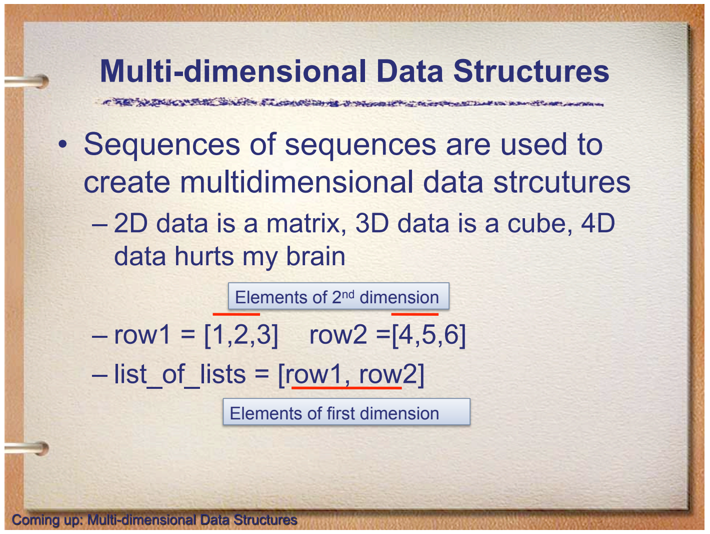
Coming up: String?



String?

- String (CDT or primitive?)
  - non-atomic: can be broken down into simpler components (characters)
  - Acts like a single value (not a structure) most of the time
  - Officially it is a CDT, but it does act like a primitive a lot

Coming up: Multi-dimensional Data Structures



Multi-dimensional Data Structures

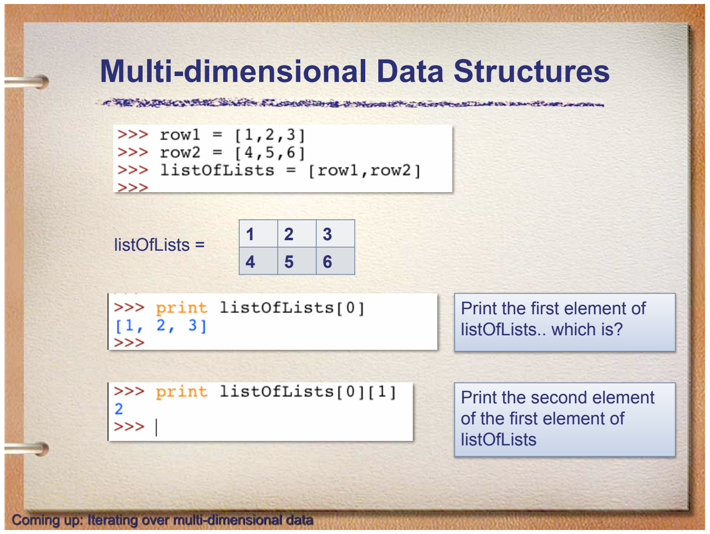
- Sequences of sequences are used to create multidimensional data strcutures
  - 2D data is a matrix, 3D data is a cube, 4D data hurts my brain

Elements of 2nd dimension

  - row1 = [1,2,3]   row2 =[4,5,6]
  - list_of_lists = [row1, row2]

Elements of first dimension

Coming up: Multi-dimensional Data Structures



Multi-dimensional Data Structures

```
>>> row1 = [1,2,3]
>>> row2 = [4,5,6]
>>> listOfLists = [row1,row2]
>>>
```

listOfLists =

1	2	3
4	5	6

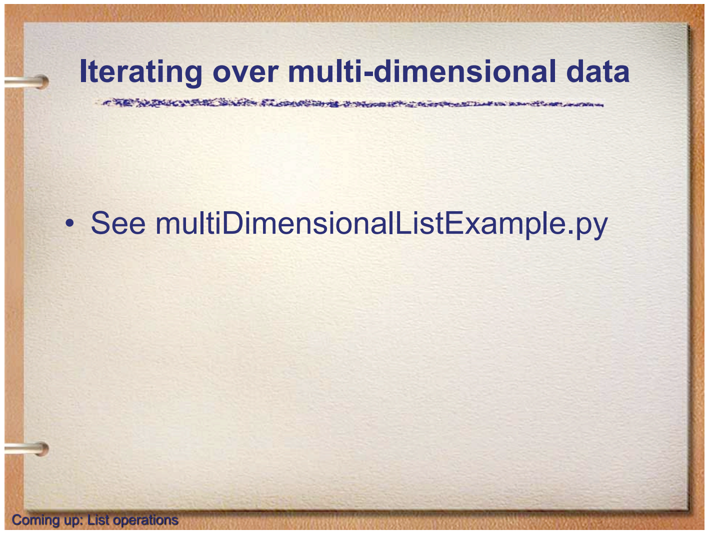
```
>>> print listOfLists[0]
[1, 2, 3]
>>>
```

Print the first element of listOfLists.. which is?

```
>>> print listOfLists[0][1]
2
>>> |
```

Print the second element of the first element of listOfLists

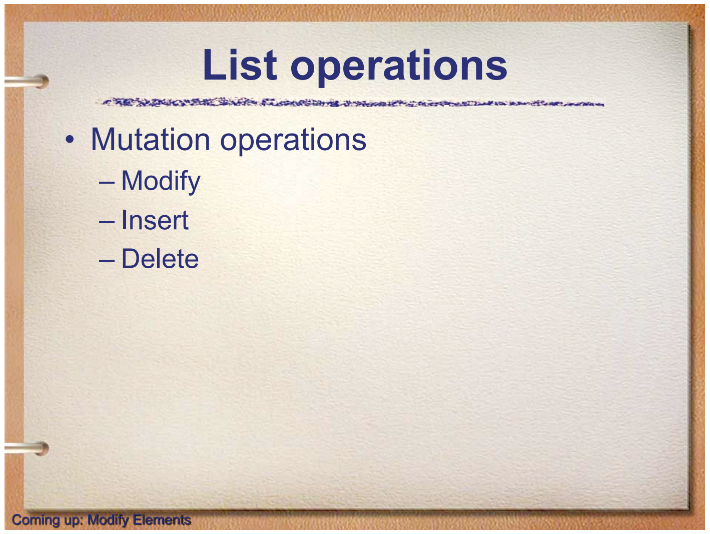
Coming up: Iterating over multi-dimensional data



Iterating over multi-dimensional data

- See multiDimensionalListExample.py

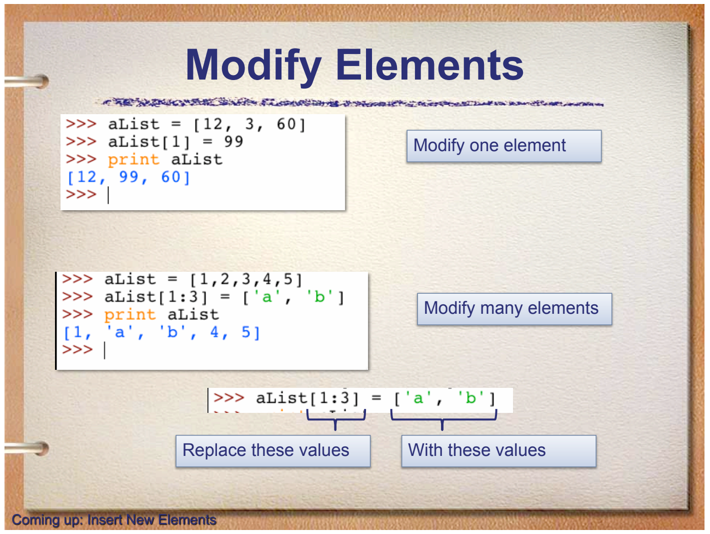
Coming up: List operations



List operations

- Mutation operations
  - Modify
  - Insert
  - Delete

Coming up: Modify Elements



Modify Elements

```
>>> aList = [12, 3, 60]
>>> aList[1] = 99
>>> print aList
[12, 99, 60]
>>> |
```

Modify one element

```
>>> aList = [1,2,3,4,5]
>>> aList[1:3] = ['a', 'b']
>>> print aList
[1, 'a', 'b', 4, 5]
>>> |
```

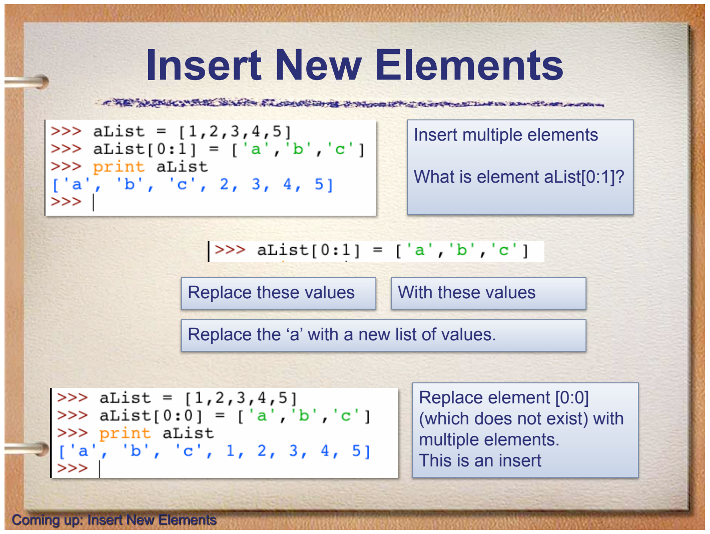
Modify many elements

```
>>> aList[1:3] = ['a', 'b']
```

Replace these values

With these values

Coming up: Insert New Elements



Insert New Elements

```
>>> aList = [1,2,3,4,5]
>>> aList[0:1] = ['a','b','c']
>>> print aList
['a', 'b', 'c', 2, 3, 4, 5]
>>> |
```

Insert multiple elements

What is element aList[0:1]?

```
>>> aList[0:1] = ['a','b','c']
```

Replace these values

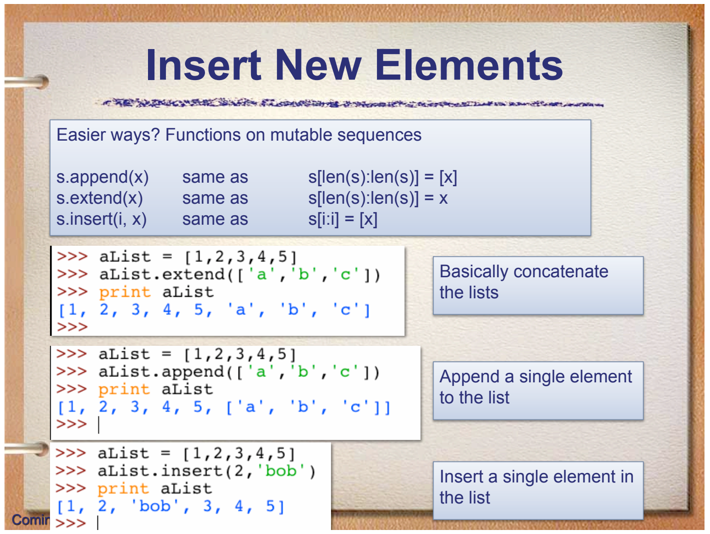
With these values

Replace the 'a' with a new list of values.

```
>>> aList = [1,2,3,4,5]
>>> aList[0:0] = ['a','b','c']
>>> print aList
['a', 'b', 'c', 1, 2, 3, 4, 5]
>>> |
```

Replace element [0:0] (which does not exist) with multiple elements.
This is an insert

Coming up: Insert New Elements



Insert New Elements

Easier ways? Functions on mutable sequences

s.append(x)	same as	s[len(s):len(s)] = [x]
s.extend(x)	same as	s[len(s):len(s)] = x
s.insert(i, x)	same as	s[i:i] = [x]

```
>>> aList = [1,2,3,4,5]
>>> aList.extend(['a','b','c'])
>>> print aList
[1, 2, 3, 4, 5, 'a', 'b', 'c']
>>>
```

Basically concatenate the lists

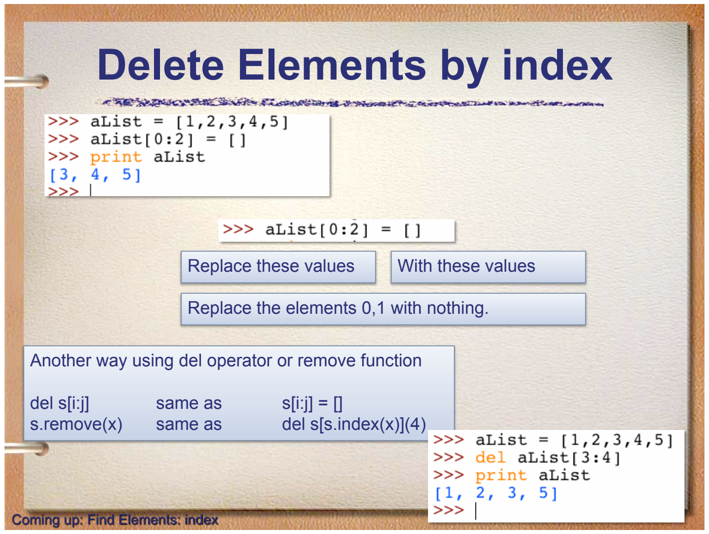
```
>>> aList = [1,2,3,4,5]
>>> aList.append(['a','b','c'])
>>> print aList
[1, 2, 3, 4, 5, ['a', 'b', 'c']]
>>> |
```

Append a single element to the list

```
>>> aList = [1,2,3,4,5]
>>> aList.insert(2,'bob')
>>> print aList
[1, 2, 'bob', 3, 4, 5]
>>> |
```

Insert a single element in the list

Comi



Delete Elements by index

```
>>> aList = [1,2,3,4,5]
>>> aList[0:2] = []
>>> print aList
[3, 4, 5]
>>> |
```

```
>>> aList[0:2] = []
```

Replace these values

With these values

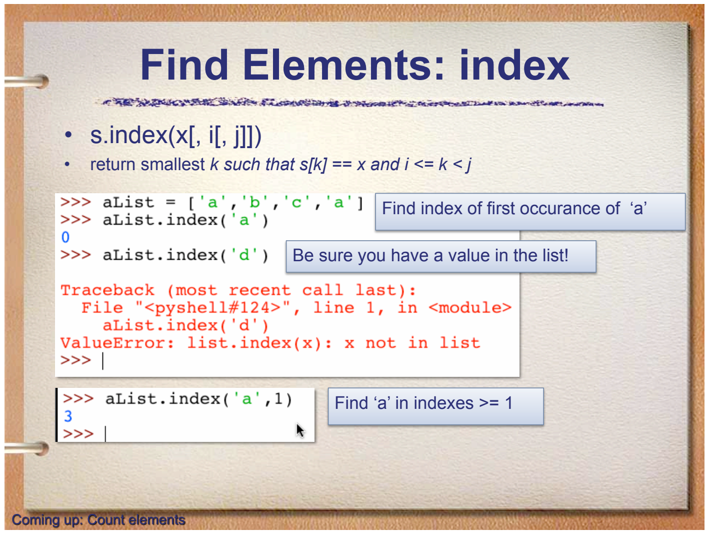
Replace the elements 0,1 with nothing.

Another way using del operator or remove function

del s[i:j]	same as	s[i:j] = []
s.remove(x)	same as	del s[s.index(x)](4)

```
>>> aList = [1,2,3,4,5]
>>> del aList[3:4]
>>> print aList
[1, 2, 3, 5]
>>> |
```

Coming up: Find Elements: index



Find Elements: index

- s.index(x[, i[, j]])
- return smallest *k such that s[k] == x and i <= k < j*

```
>>> aList = ['a','b','c','a']
>>> aList.index('a')
0
>>> aList.index('d')

Traceback (most recent call last):
  File "<pyshell#124>", line 1, in <module>
    aList.index('d')
ValueError: list.index(x): x not in list
>>> |
```

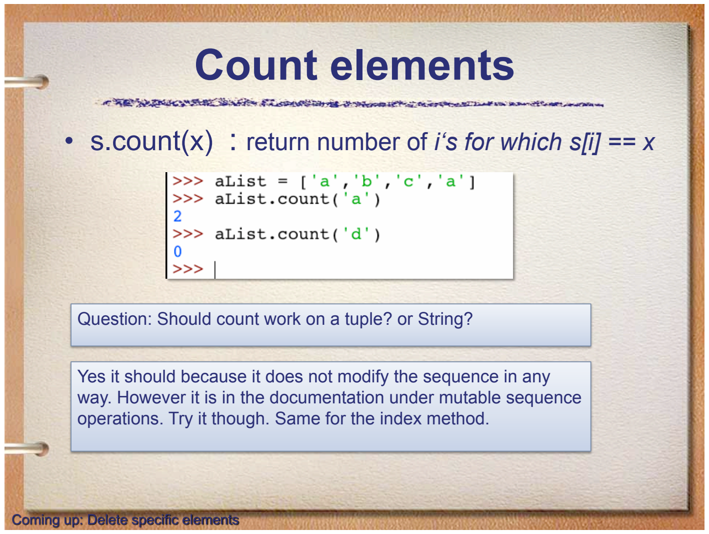
Find index of first occurance of 'a'

Be sure you have a value in the list!

```
>>> aList.index('a',1)
3
>>> |
```

Find 'a' in indexes >= 1

Coming up: Count elements



Count elements

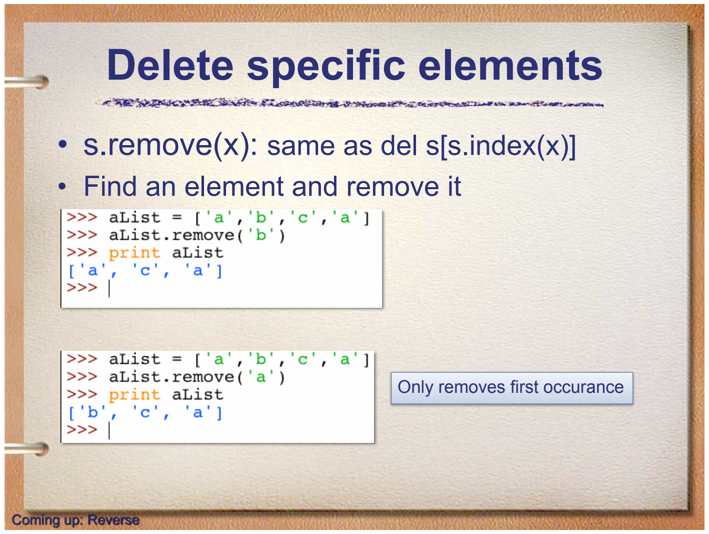
- s.count(x) : return number of *i's for which s[i] == x*

```
>>> aList = ['a','b','c','a']
>>> aList.count('a')
2
>>> aList.count('d')
0
>>> |
```

Question: Should count work on a tuple? or String?

Yes it should because it does not modify the sequence in any way. However it is in the documentation under mutable sequence operations. Try it though. Same for the index method.

Coming up: Delete specific elements



Delete specific elements

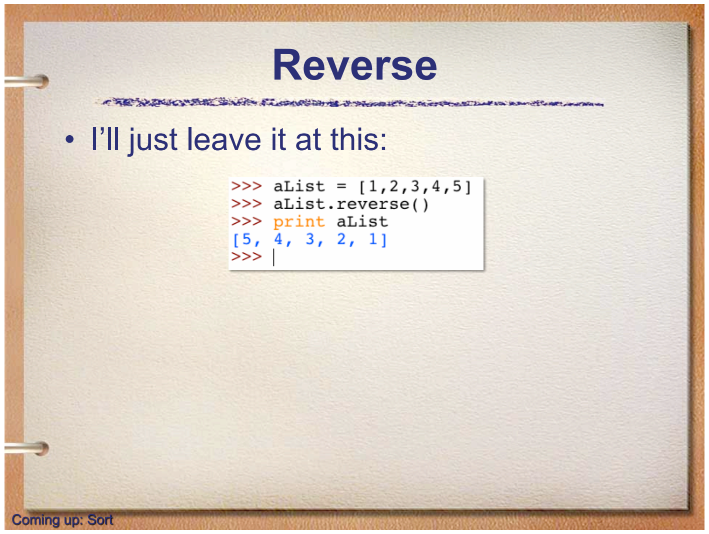
- s.remove(x): same as del s[s.index(x)]
- Find an element and remove it

```
>>> aList = ['a','b','c','a']
>>> aList.remove('b')
>>> print aList
['a', 'c', 'a']
>>> |
```

```
>>> aList = ['a','b','c','a']
>>> aList.remove('a')
>>> print aList
['b', 'c', 'a']
>>> |
```

Only removes first occurance

Coming up: Reverse

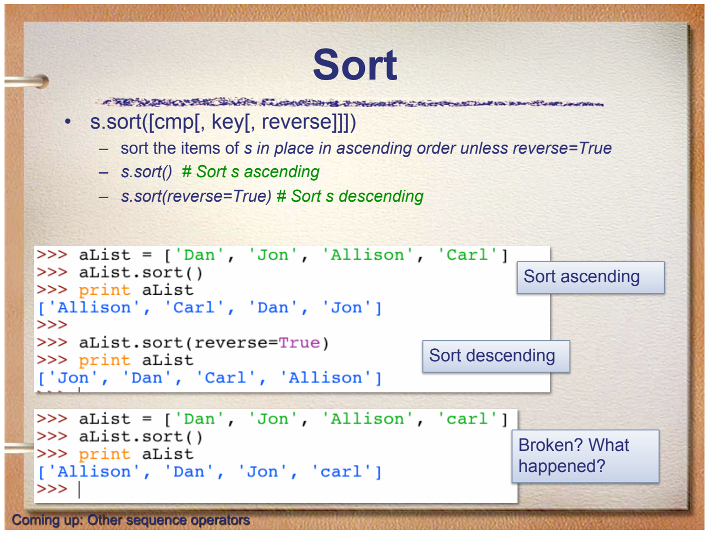


Reverse

- I'll just leave it at this:

```
>>> aList = [1,2,3,4,5]
>>> aList.reverse()
>>> print aList
[5, 4, 3, 2, 1]
>>> |
```

Coming up: Sort



Sort

- s.sort([cmp[, key[, reverse]]])
  - sort the items of *s in place in ascending order unless reverse=True*
  - *s.sort() # Sort s ascending*
  - *s.sort(reverse=True) # Sort s descending*

```
>>> aList = ['Dan', 'Jon', 'Allison', 'Carl']
>>> aList.sort()
>>> print aList
['Allison', 'Carl', 'Dan', 'Jon']
>>>
>>> aList.sort(reverse=True)
>>> print aList
['Jon', 'Dan', 'Carl', 'Allison']
```

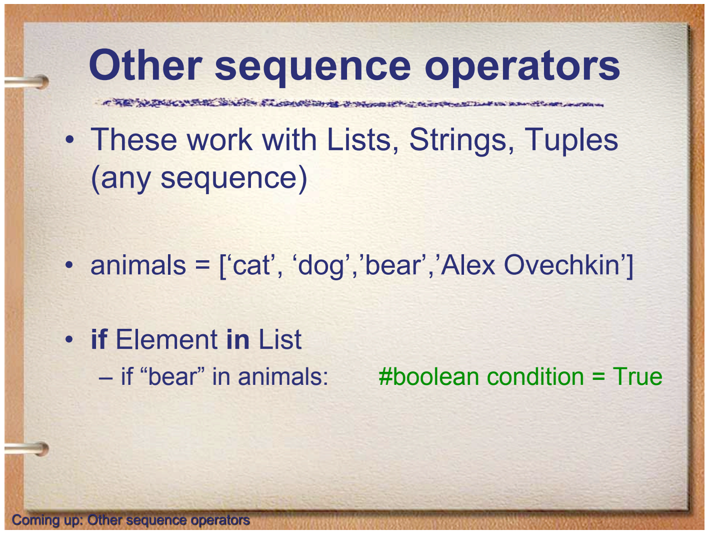
Sort ascending

Sort descending

```
>>> aList = ['Dan', 'Jon', 'Allison', 'carl']
>>> aList.sort()
>>> print aList
['Allison', 'Dan', 'Jon', 'carl']
>>> |
```

Broken? What happened?

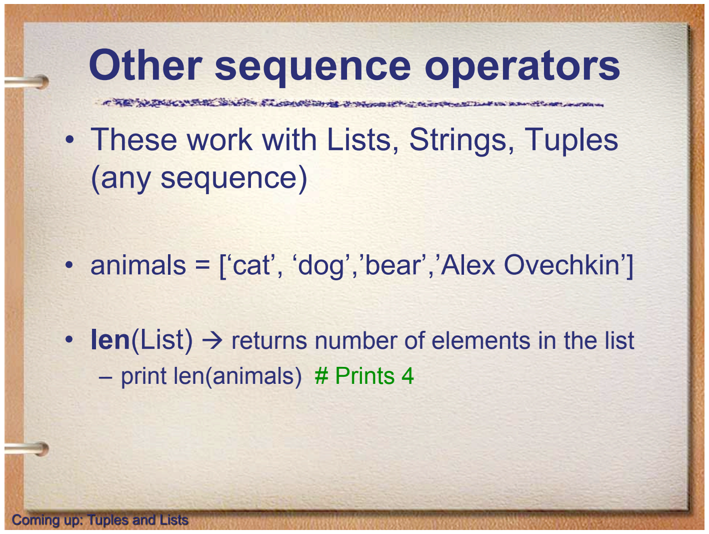
Coming up: Other sequence operators



Other sequence operators

- These work with Lists, Strings, Tuples (any sequence)
- animals = ['cat', 'dog','bear','Alex Ovechkin']
- **if** Element **in** List
  - if "bear" in animals: #boolean condition = True

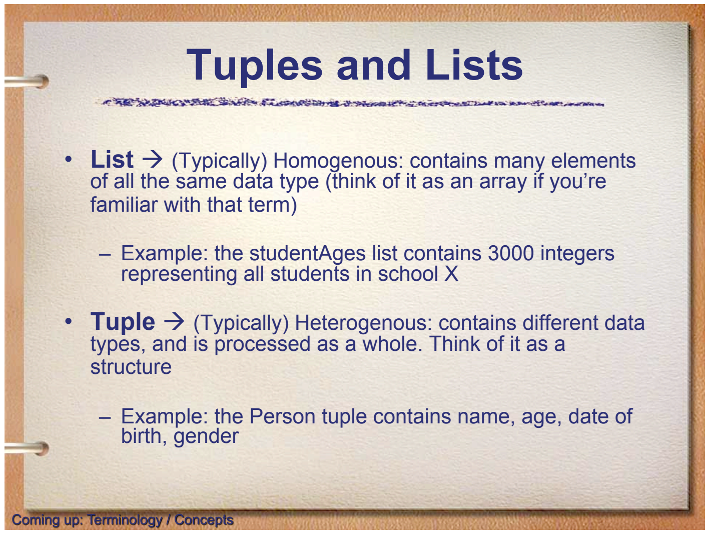
Coming up: Other sequence operators



Other sequence operators

- These work with Lists, Strings, Tuples (any sequence)
- animals = ['cat', 'dog','bear','Alex Ovechkin']
- **len**(List) → returns number of elements in the list
  - print len(animals) # Prints 4

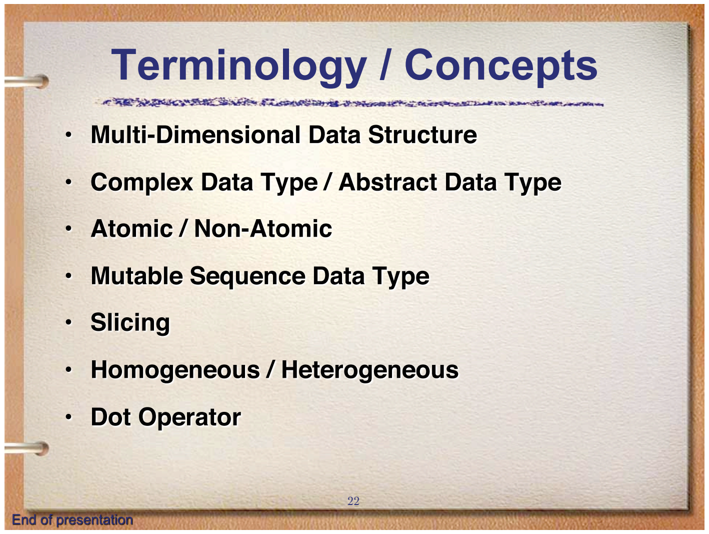
Coming up: Tuples and Lists



Tuples and Lists

- **List** → (Typically) Homogenous: contains many elements of all the same data type (think of it as an array if you're familiar with that term)
  – Example: the studentAges list contains 3000 integers representing all students in school X
- **Tuple** → (Typically) Heterogenous: contains different data types, and is processed as a whole. Think of it as a structure
  – Example: the Person tuple contains name, age, date of birth, gender

Coming up: Terminology / Concepts



Terminology / Concepts

- **Multi-Dimensional Data Structure**
- **Complex Data Type / Abstract Data Type**
- **Atomic / Non-Atomic**
- **Mutable Sequence Data Type**
- **Slicing**
- **Homogeneous / Heterogeneous**
- **Dot Operator**

22

End of presentation