Introduction to Python
Topics

Christopher Barker

UW Continuing Education

October 15, 2013

Table of Contents

Review of Previous Class

- .
- .
- .

Lightning Talks

Lightning talks today:

Homework review

Homework Questions?

My Solution

topic

Some Stuff

```
sample code
```

Lists

Lists Literals

```
>>> []
[]
>>> list()
[]
>>> [1, 2, 3]
[1, 2, 3]
>>> [1, 3.14, "abc"]
[1, 3.14, 'abc']
```

List Indexing

Indexing just like all sequences

```
>>> food = ['spam', 'eggs', 'ham']
>>> food[2]
'ham'
>>> food[0]
'spam'
>>> food[42]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

List Mutability

Lists are mutable

```
>>> food = ['spam', 'eggs', 'ham']
>>> food[1] = 'raspberries'
>>> food
['spam', 'raspberries', 'ham']
```

List Elements

Each element is a value, and can be in multiple lists and have multiple names (or no name)

```
>>> name = 'Brian'
 >>> a = [1, 2, name]
 >>> b = [3, 4, name]
 >>> name
 'Brian'
 >>> a
 [1, 2, 'Brian']
 >>> b
 [3, 4, 'Brian']
 >>> a[2]
 'Brian'
 >>> b[2]
 'Brian'
```

List Methods

`.append(), .insert()`

```
>>> food = ['spam', 'eggs', 'ham']
>>> food.append('sushi')
>>> food
['spam', 'eggs', 'ham', 'sushi']
>>> food.insert(0, 'carrots')
>>> food
['carrots', 'spam', 'eggs', 'ham', 'sushi']
```

List Methods

```
.extend()
```

```
>>> food = ['spam', 'eggs', 'ham']
>>> food.extend(['fish', 'chips'])
>>> food
['spam', 'eggs', 'ham', 'fish', 'chips']
```

could be any sequence:

```
>>>  food
>>>  ['spam', 'eggs', 'ham']
>>>  silverware = ('fork', 'knife', 'spoon') # a tuple
>>>  food.extend(silverware)
>>>  food
>>>  ['spam', 'eggs', 'ham', 'fork', 'knife', 'spoon']
```

List Methods

`pop()`, `remove()`

```
In [203]: food = ['spam', 'eggs', 'ham', 'toast']
In [204]: food.pop()
Out[204]: 'toast'
In [205]: food.pop(0)
Out[205]: 'spam'
In [206]: food
Out[206]: ['eggs', 'ham']
In [207]: food.remove('ham')
In [208]: food
Out[208]: ['eggs']
```

List Constructor

`list()` accepts any sequence and returns a list of that sequence

```
>>> word = 'Python '
>>> chars = []
>>> for char in word:
...     chars.append(char)
>>> chars
['P', 'y', 't', 'h', 'o', 'n', ' ']
>>> list(word)
['P', 'y', 't', 'h', 'o', 'n', ' ']
```

String to List to String

If you need to change individual letters... you can do this, but usually `somestring.replace()` will be enough

```
In [216]: name = 'Chris'
In [217]: lname = list(name)
In [218]: lname[0:2] = 'K'
In [219]: name = ''.join(lname)
In [220]: name
Out[220]: 'Kris'
```

Building up strings in a list

```
In [221]: msg = []

In [222]: msg.append('The first line of a message')

In [223]: msg.append('The second line of a message')

In [224]: msg.append('And one more line')

In [225]: print '\n'.join(msg)
The first line of a message
The second line of a message
And one more line
```

List Slicing

Slicing makes a copy

```
In [227]: food = ['spam', 'eggs', 'ham', 'sushi']

In [228]: some_food = food[1:3]

In [229]: some_food[1] = 'bacon'

In [230]: food
Out[230]: ['spam', 'eggs', 'ham', 'sushi']

In [231]: some_food
Out[231]: ['eggs', 'bacon']
```

List Slicing

Easy way to copy a whole list

```
In [232]: food
Out[232]: ['spam', 'eggs', 'ham', 'sushi']

In [233]: food2 = food[:]

In [234]: food is food2
Out[234]: False
```

but the copy is “shallow”:
http://docs.python.org/library/copy.html

List Slicing

“Shallow” copy

```
In [249]: food = ['spam', ['eggs', 'ham']]
In [251]: food_copy = food[:]
In [252]: food[1].pop()
Out[252]: 'ham'
In [253]: food
Out[253]: ['spam', ['eggs']]
In [256]: food.pop(0)
Out[256]: 'spam'
In [257]: food
Out[257]: [['eggs']]
In [258]: food_copy
Out[258]: ['spam', ['eggs']]
```

Name Binding

Assigning to a name does not copy:

```
>>> food = ['spam', 'eggs', 'ham', 'sushi']
>>> food_again = food
>>> food_copy = food[:]
>>> food.remove('sushi')
>>> food
['spam', 'eggs', 'ham']
>>> food_again
['spam', 'eggs', 'ham']
>>> food_copy
['spam', 'eggs', 'ham', 'sushi']
```

List Iterating

Iterating over a list

```
>>> food = ['spam', 'eggs', 'ham', 'sushi']
>>> for x in food:
...   print x
...
spam
eggs
ham
sushi
```

Processing Lists

A common pattern

```
filtered = []
for x in somelist:
    if should_be_included(x):
        filtered.append(x)
del(somelist)  # maybe
```

you don't want to be deleting items from the list while iterating...

Mutating Lists

if you're going to change the list, iterate over a copy for safety

```
>>> food = ['spam', 'eggs', 'ham', 'sushi']
>>> for x in food[:]:
   ...    # change the list somehow
   ...
```

insidious bugs otherwise

operators vs methods

What's the difference?

```
>>> food = ['spam', 'eggs', 'ham']
>>> more = ['fish', 'chips']
>>> food = food + more
>>> food
['spam', 'eggs', 'ham', 'fish', 'chips']

>>> food = ['spam', 'eggs', 'ham']
>>> more = ['fish', 'chips']
>>> food.extend(more)
>>> food
['spam', 'eggs', 'ham', 'fish', 'chips']
```

(the operator makes a new list...)

in

```
>>> food = ['spam', 'eggs', 'ham']
>>> 'eggs' in food
True
>>> 'chicken feet' in food
False
```

reverse()

```
>>> food = ['spam', 'eggs', 'ham']
>>> food.reverse()
>>> food
['ham', 'eggs', 'spam']
```

sort()

```
>>> food = ['spam', 'eggs', 'ham', 'sushi']
>>> food.sort()
>>> food
['eggs', 'ham', 'spam', 'sushi']
```

note:

```
>>> food = ['spam', 'eggs', 'ham', 'sushi']
>>> result = food.sort()
>>> print result
None
```

Sorting

How should this sort?

```
>>> s
[[2, 'a'], [1, 'b'], [1, 'c'], [1, 'a'], [2, 'c']]
```

Sorting

How should this sort?

```
>>> s
[[2, 'a'], [1, 'b'], [1, 'c'], [1, 'a'], [2, 'c']]

>>> s.sort()
>>> s
[[1, 'a'], [1, 'b'], [1, 'c'], [2, 'a'], [2, 'c']]
```

Sorting

You can specify your own compare function:

```
In [279]: s = [[2, 'a'], [1, 'b'], [1, 'c'], [1, 'a'], [2,
In [281]: def comp(s1,s2):
   .....:     if s1[1] > s2[1]: return 1
   .....:     elif s1[1]<s2[1]: return -1
   .....:     else:
   .....:         if s1[0] > s2[0]: return 1
   .....:         elif s1[0] < s2[0]: return -1
   .....:     return 0
In [282]: s.sort(comp)
In [283]: s
Out[283]: [[1, 'a'], [2, 'a'], [1, 'b'], [1, 'c'], [2, 'c'
```

Sorting

Mixed types can be sorted.

“objects of different types always compare unequal, and are ordered consistently but arbitrarily.”

`http://docs.python.org/reference/expressions.html#not-in`

Searching

Finding or Counting items

```
In [288]: l = [3,1,7,5,4,3]

In [289]: l.index(5)
Out[289]: 3

In [290]: l.count(3)
Out[290]: 2
```

List Performance

- indexing is fast and constant time: O(1)
- x in s proportional to n: O(n)
- visiting all is proportional to n: O(n)
- operating on the end of list is fast and constant time: O(1)
  append(), pop()
- operating on the front (or middle) of the list depends on n: O(n)
  pop(0), insert(0, v)
  But, reversing is fast. Also, collections.deque

`http://wiki.python.org/moin/TimeComplexity`

Lists vs. Tuples

List or Tuples

If it needs to mutable: list

If it needs to be immutable: tuple
(dict key, safety when passing to a function)

Otherwise ... taste and convention

List vs Tuple

Convention:

Lists are Collections (homogeneous):

– contain values of the same type
– simplifies iterating, sorting, etc

tuples are mixed types:

– Group multiple values into one logical thing – Kind of like simple C structs.

List vs Tuple

- Do the same operation to each element?
- Small collection of values which make a single logical item?
- To document that these values won't change?
- Build it iteratively?
- Transform, filter, etc?

List vs Tuple

- Do the same operation to each element? **list**
- Small collection of values which make a single logical item? **tuple**
- To document that these values won't change? **tuple**
- Build it iteratively? **list**
- Transform, filter, etc? **list**

List Docs

The list docs:

`http://docs.python.org/library/stdtypes.html#mutable-sequence-types`

(actually any mutable sequence....)

tuples and commas..

Tuples don't NEED parentheses...

```
In [161]: t = (1,2,3)
In [162]: t
Out[162]: (1, 2, 3)

In [163]: t = 1,2,3
In [164]: t
Out[164]: (1, 2, 3)

In [165]: type(t)
Out[165]: tuple
```

tuples and commas..

Tuples do need commas...

```
In [156]: t = ( 3 )

In [157]: type(t)
Out[157]: int

In [158]: t = (3,)
In [159]: t
Out[159]: (3,)

In [160]: type(t)
Out[160]: tuple
```

LAB

List Lab

`week-03/code/list_lab.rst`

LAB

Some lab excercises

Lightning Talk

Lightning Talks:

person 1

person 2

Homework

Recommended Reading:

- some stuff

Do:

- Some things