Python Lists and for Loops

CMSC 201
Fall 2012
Instructor: John Park
Lecture Section 01
Discussion Sections 02-08, 16, 17

Learning Outcomes

• Be aware that multiple items can be stored in a list.
• Become acquainted with Python’s built-in functions, methods and operators that can be used for lists.
• Know the syntax of working with lists and items in a list.
• Understand that for loops are used to iterate over sequences, including lists.

What’s a List

• Definition:
  – A list is an ordered collection of items, where the items can be of any type.
  – Each item has a numerical index, indicating the position in the list, which is what is meant by "ordered". Indices start at zero.
• Operations:
  – adding items to the list
  – removing items from the list
  – finding items in the list
  – indexing (accessing or replacing an item in the list).
Lists vs. Arrays

- A list in Python is similar to, but not the same as, an array in C or Java, or other languages; there are some significant differences.
- Although you may have arrays of any type, arrays are homogeneous, meaning they can only hold data of one type. Python lists are heterogeneous. The type of the data being stored can be different at different indices in the same list.
- Arrays are a fixed size, but lists shrink and grow as necessary to hold the exact number of items being stored.

Indexing

- Indexing of lists in Python is similar to indexing of arrays.
- Indices begin with 0. For a list of length n, the last index will be n - 1.
- Here’s how we can picture a list of length 5, showing indices of 0 through 4.

    0:  1:  2:  3:  4:

- Python allows us to use negative indices into a list, as well as positive ones.
- The index -1 can be used to access the last item in the list. The next to the last item can be accessed by using the index, -2, etc.
Lists in Python

- There are several ways to create new lists:

```python
>>> # assign the literal empty list
>>> empty = []
>>> empty
[]

>>> # use the list constructor
>>> empty = list()
>>> empty
[]

>>> # give a literal assignment of items
>>> items = ['bread', 'milk', 'cheese', 'cider']
>>> items
['bread', 'milk', 'cheese', 'cider']
```

- ... and more ways to create new lists:

```python
>>> # use the list constructor with a string
>>> letters = list("hello")
>>> letters
['h', 'e', 'l', 'l', 'o']

>>> # use split which returns a list of strings
>>> input = "a bunch of words"
>>> words = input.split()
>>> words
['a', 'bunch', 'of', 'words']
```

Operators

Indexing:
```
list[index] - get or set an item at the given index in the list
```

```python
>>> items[2]
'cheese'
>>> items[3]
'cider'
>>> items[3] = "lemonade"
>>> items[3]
'lemonade'
```
Operators

Containment:
• item in list - see if the item is in the list
  >>> "milk" in items
  True
  >>> "juice" in items
  False

Operators

Equality:
• list1 == list2 - see if two lists are the same
  >>> "two words".split() == ["two", "words"]
  True
  >>> ["this", "that"] == ["that", "this"]
  False

Operating on Lists

There are 3 ways of operating on lists:
• Indexing
  – Use [] notation, like array references
• Built-in functions
• Methods:
  – peek ahead at object-oriented programming
  – [Called by prepending variable name to function call]
    (more accurate language: “called by invoking the method on the object”)
Fancy Indexing: Slices

- You can specify a fragment of a list, or slice, by giving two numbers in the index (let's say $m$ and $n$); the new list will include the elements of the original list from index $m$ through $n-1$
- If $m$ is left out, it implies the start of the list
- If $n$ is left out, it implies the end of the list
- E.g.: `items[2:4]`

```
0: 1: 2: 3: 4: 5:
```

Built-in Functions

Built-in functions that operate on lists:
- `len(list)` - count number of items in the list
- `del(list[index])` - remove the item at index from the list
  (This syntax for `del()` is a little unorthodox)

Methods

- `append(item)` - add the item to end of the list
- `count(item)` - count occurrences of the item in the list
- `extend(list)` - append multiple items to the end of the list
- `index(item)` - locate the index of an item in the list
Methods

- `insert(index, item)` - insert an item before the one at the index
- `remove(item)` - remove the first occurrence of the item in the list
- `reverse()` - reverse the order of the items in the list
- `sort()` - sort the list

Examples—Built-in Functions

- `len(list)` - count number of items in the list

```python
>>> items
['bread', 'cheese', 'eggs', 'ham', 'lemonade', 'lettuce', 'milk', 'tomatoes']
>>> len(items)
8
```

Examples—Built-in Functions

- `del(list[index])` - remove the item at the index from the list

```python
>>> items
['bread', 'cheese', 'eggs', 'ham', 'lemonade', 'lettuce', 'milk', 'tomatoes']
>>> del(items[4])
>>> items
['bread', 'cheese', 'eggs', 'ham', 'lettuce', 'milk', 'tomatoes']
```
Examples—List Methods

• append(item) - add an item to the end of the list

```python
>>> items
['bread', 'milk', 'cheese', 'lemonade']
>>> items.append('ham')
>>> items
['bread', 'milk', 'cheese', 'lemonade', 'ham']
>>> items.append('ham')
>>> items
['bread', 'milk', 'cheese', 'lemonade', 'ham', 'ham']
```
Examples—List Methods

• index(item) - locate the index of an item in the list

```python
>>> items
['bread', 'milk', 'cheese', 'lemonade',
'ham', 'lettuce', 'tomatoes']
>>> items.index('milk')
1
```
Examples—List Methods

• reverse() - reverse the order of the items in the list; NB: it does it in place!

```python
>>> items
['bread', 'milk', 'eggs', 'cheese', 'lemonade', 'ham', 'lettuce', 'tomatoes']
>>> items.reverse()

>>> items
['tomatoes', 'lettuce', 'ham', 'lemonade', 'cheese', 'eggs', 'milk', 'bread']
```

Examples—List Methods

• sort() - sort the list, again in-place

```python
>>> items
['tomatoes', 'lettuce', 'ham', 'lemonade', 'cheese', 'eggs', 'milk', 'bread']
>>> items.sort()

>>> items
['bread', 'cheese', 'eggs', 'ham', 'lemonade', 'lettuce', 'milk', 'tomatoes']
```

Using Lists

• Use a list when order matters.
• Items stay in the order they're inserted in with append()
• A list is indexed in order
• A list can also be changed so that the items are in sorted order.
A Simple Python for-loop

• C/C++, Java, Perl, and many other languages have a convenience form for looping over a range of values:
  ```c
  for (i = 0; i < 10; i++) {
    printf("%d ", i);
  }
  ```
• Here is an equivalent Python for-loop that prints out the integers from 0 through 9:
  ```python
  for i in range(10):
    print i,
  ```
• However, Python for-loops can do much more

Iterating Over Sequences

• `for` loops can iterate over lists, strings and other sequences.
• We can iterate over elements in a list:
  ```python
  >>> myList = [32, 45, 34, 76, 45]
  >>> sum = 0.0
  >>> for n in myList:
  ...      sum += n
  ...
  >>> print sum / len(myList)
  46.4
  ```

Python’s range() Function

• In actuality, Python’s `range()` function actually just generates a list, from 0 to n - 1:
  ```python
  >>> numbers = range(5)
  >>> numbers
  [0, 1, 2, 3, 4]
  >>> type(numbers)
  <type 'list'>
  >>> print range(2, 5)
  [2, 3, 4]
  >>> print range(2, 10, 2)
  [2, 4, 6, 8]
  >>> print range(5, 1, -1)
  [5, 4, 3, 2]
  ```
Example Using a List

The following example shows:

• Placing your code in main()
• Building a list, beginning with an empty list, using append
• Iteration through an existing list using "in"
  — Allows us to traverse lists more efficiently than using a combination of range() and indexing ([[])

```python
def main():
    # Begin with an empty list
    gpas = []
    # Builds a list of 10 GPAs
    for i in range(10):
        gpa = input("Enter a student's GPA : ")
        gpas.append(gpa)
    # Initialize the accumulator
    total = 0.0
    # Can use "in" instead of range
    for gpa in gpas:
        # Accumulate into total
        total += gpa
    # Calculate average (total is already a float)
    average = total / len(gpas)
    print "The average GPA of this section is ", average
main()
```

Lists in Python

• Here's the output:

```
linuxserver1.cs.umbc.edu python list.py
Enter a student's GPA : 3.0
Enter a student's GPA : 3.5
Enter a student's GPA : 2.0
Enter a student's GPA : 2.5
Enter a student's GPA : 1.5
Enter a student's GPA : 4.0
Enter a student's GPA : 2.7
Enter a student's GPA : 2.8
Enter a student's GPA : 3.2
Enter a student's GPA : 3.4
The average GPA of this section is 2.86
```

```
An Improved List Example

Add the following improvements to the code:

• The user needs to understand what the program is doing, so it’s important that you always print a greeting.

• An important step in coding is to make sure the data structure you are using (in this case a list), gets populated (in this case with the data being entered by the user).
  – Do this by adding a debugging statement that dumps out the values stored in the GPA list

```python
def main():
    # Tell the user what the program will do - Give a Greeting
    print "This program will find the average GPA for a section of 20!"
    print
    # Begin with an empty list
    gpas = []
    # Builds a list of 20 GPAs
    for i in range(20):
        gpa = input("Enter a student's GPA : ")
        gpas.append(gpa)
    # Mandatory Debugging Step (tmp) - Make sure you really made a list!
    for i in range(20):
        print "gpas[", i, "] = ", gpas[i]
    # Initialize the accumulator
    total = 0.0
    # Can use "in" instead of range
    for gpa in gpas:
        # Accumulate into total
        total += gpa
    # Calculate average (total is already a float)
    average = total / len(gpas)
    # Add a blank line to separate the input from the result
    print "The average GPA of this section is ", average
main()```

List Exercise

In groups of 3 or 4, write Python code that will:

• get 10 integers from the user and store them in a list
• find the total of the numbers in the list
• find the minimum value in the list
• find the maximum value in the list
• find the average of the values in the list
• print the total, minimum, maximum & average similar to the following sample output:
List Exercise

• Here’s the output:

<table>
<thead>
<tr>
<th>Total</th>
<th>115</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>1</td>
</tr>
<tr>
<td>Maximum</td>
<td>25</td>
</tr>
<tr>
<td>Average</td>
<td>11.5</td>
</tr>
</tbody>
</table>

• Your code should be in main()
• Check to make sure your list is populated using a mandatory debugging statement