Introduction to:
Computers & Programming:
Collections of Objects:
Lists, Tuples, Dictionaries, Sets

Adam Meyers

New York University

Outline

- Data Structures consisting of different types of objects
- Sequences
  - Lists
  - Tuples (hardly used)
- Unordered collections of objects
  - Dictionaries
  - Sets
- Revisiting the Some Day in the Future Problem

Sequences

- Ranges: consist of numbers
  - range(1,5)
- Strings: consist of characters
  - 'Hello World'
- Lists and Tuples consist of anything
  - Lists
    - [1,range(5),'the book',57,'dog',[1,2,4]]
  - Tuples
    - (1,range(5),'the book',57,'dog',[1,2,4])

Mutability

- ranges, tuples, strings (and some sets) cannot be changed
  - once created, you can copy them, but you can't change them
- Lists, dictionaries (and some sets) can be modified
- Examples:

```
mixed_list = [1,range(5),'the book',57,'dog',[1,2,4]]
mixed_list[1] = 'abc'
```

  - You cannot do that with mixed_tuple, defined the same way

Mutability 2

- You can't really change strings, but you can copy them
  - When we iterate, we are copying strings

```
string = ''
for character in 'hello':
    string = string + character + '-'
## We are copying here
```

- We could copy tuples as well, but tuples are rarely used. Most python programmers just use lists

Applying What We Know About Sequences

- Indices
  - sequence[index]
- Slices
  - sequence[start:end]
- Other Functions/Operators
  - len(sequence) – length
  - item in sequence – tests set membership
  - for item in sequence: ## loops

    … block-of-text ..

Functions can Change Mutible Objects

- This might seem odd since functions cannot change global variables
  - Because variables are passed by reference
- It is possible to things about an object. A variable pointing to that object points to it regardless of whether or not it changes.
  - Think about variables pointing to turtles. The position of a turtle is a property that can change, but variables pointing to a turtle still point to it.

Example of a Function that Changes a List

- def change_list(list, index, new_item):

  list[index] = new_item
- def experiment_with_list():

  list = [1,2,3,4,5]

  print('1st version: ',list)

  change_list(list,3,'three')

  print('2nd version: ',list)

  return(list)

Lists Can be Self-Referential

- Self_Referential_List = [1, 2, 3]
- Self_Referential_List[0] = Self_Referential_List

```
>>> self_referential_list
[[...], 2, 3]
>>> self_referential_list[0]
[[...], 2, 3]
>>> self_referential_list[0][0]
[[...], 2, 3]
```

- It is probably best to avoid this situation

Tuple/List Specific functions

- X.count(item) ## counts times that item occurs in tuple/list
- X.index(item) ## location of first occurrence of item in tuple/list

```
>>> duck_list = ['duck','duck','duck','goose']
>>> duck_list.count('duck')
3
>>>  duck_list.index('duck')
0
>>>
```

List Specific Functions

- list.append(x) ## adds x to the end of the list
  - this changes the list
- Other functions that change lists: extend, insert, pop, remove, reverse, sort
- Examples
  - mixed_list.append('abc')
  - mixed_list.extend(['a','b','c'])
  - mixed_list.pop(2)
  - mixed_list.remove('a')
  - mixed_list.reverse()

Sort puts strings in ascending order

- Lists of a single object type are sortable

```
>>> string_list = ['duck', 'cat','bear','dog']
>>> string_list.sort()
>>> string_list
['bear', 'cat', 'dog', 'duck']
```

  - Number_list = [15,1,68,2,1.4]
- In contrast, mixed_list cannot be ordered
  - It contains both a string and a list
  - Thus: mixed_list.sort() results in an error

Problem from Earlier in the Semester

- Given
  - Today's date:
    - Month, day, year, and day of the week
  - A number of days in the future between 1 and 1500
- How should we go about identifying that day in the future?
  - Month, day, year and day of the week
- future-date.py

Cooperative Problem for Class: Part I

1. Write a program that returns a list of 50 items consisting of potential members of the list:

   ['square 20','circle 20','triangle 20', 'left 15 40','left 70 40','left 90 40', 'straight 50']

   - Each item should be chosen randomly and added to the output list using the list.append function
     - **list.append(item)** – changes list (adds item to the end)
     - **list = list.append(item) – will not work!!!**
   - To do this, remember to import the *random* module

2. Write a turtle program, beginning the file with:

```
import turtle
import random
import math
import time
my_screen = turtle.Screen()
my_turtle = turtle.Turtle()
```

Problem for Class Part II

3. Write a function that takes as input a list of strings and loops through the list, causing the turtle to do a different action based on the string. (Assume pen is always down)
   - 'square 20': square with each side 20 long
   - 'circle 20': circle with radius of 20
     - turtle.circle(20)
   - 'triangle 20': equilateral triangle with sides 20 long
   - 'left num1 num2':
     - turtle.left(num1)
     - turtle.fd(num2)
   - 'straight num'
     - turtle.fd(num)
   - For First Version: we can hard code these

Problem in Class: Part III

4. Write a function that uses 1 to get instructions and then executes the instructions using 3
5. Questions to consider:
   - How could we write better versions of 1 and 3?
     - That were more flexible
     - That did not have so many things hard coded
     - Which special string functions would we have to use?
     - What are the advantages of having 1 and 3 be separate functions?

Random Sentence Generation Program

- 1000 most common words according to: http://www.bckelk.ukfsn.org/words/uk1000.html
- Sorted them automatically into: nouns, plural_nouns, adjectives,verbs,adverbs
  - According to some dictionaries that I have
  - Some criteria for simple sentences
    - simple past tense verbs
    - adverbs: manner, temporal, evaluative only

Random Sentence Generation 2

- A sentence is:
  - A noun group (Noun Phrase)
  - A verb group (Verb Phrase)
- A verb group is:
  - A verb
  - A noun group
  - An optional adverb
- A noun group is:
  - Optionally definite
  - An optional adjective
  - A singular or plural noun
    - Definite = + *the*
    - Indefinite = nothing for plurals or *a* for singular

List Comprehensions

- Comprehensions are shortcuts for creating lists
- [2 * n + 7 for n in range(1,11)]
  - Equivalent to output of the following function called with 10 as an argument:

```
def get_2_n_plus_7(length):
    output = []
    for n in range(1,length+1):
        output.append(2 * n +7)
    return(output)
```

Filtered List Comprehensions

- result = [n for n in [3,4,5,6,7] if n > 4]
  - Sets result to [5,6,7]

```
result = []
for n in [3,4,5,6,7]:
    if n >4:
        result.append(n)
```

- Of course, we can combine these two

```
[[n,2 * n+7] for n in range(8) if n%2==0]
```

Dictionaries

- A dictionary can be initialized as follows:

  my_dictionary = {'eat' : 'verb', 'book' : 'noun', 'orange' : 'adjective'}

- Items can be looked up in a dictionary, as follows:

  my_dictionary['eat']

  my_dictionary.get('eat')

  – If key is not in dictionary, these give error

- To check if item is in dictionary:

  'eat' in my_dictionary

- Items can be added or changes as follows:

  my_dictionary['book'] = 'common_noun'

Dictionaries 2

- What is a dictionary in Python?
  - Dictionaries consists of sets of key/value pairs
    - For example, the key 'book' currently has the value 'common_noun'
  - These are not ordered. We look items up by keyword, not by index as with sequences.
- Why have a dictionary?
  - To store fairly large amounts of information that has no natural order.
- Examples
  - Words: part of speech (noun, verb, etc.), definitions, pronunciation, etc. (Lexicons)
  - Sports player: statistics
  - Food: ingredients
  - Medicine: side effects
  - Telephone books (or reverse telephone books)

Dictionaries are Hash Tables

- The dictionary data structure is Python's implementation of a hash table, a popular data structure for mapping keys to values
- There is a special “hash” function which maps keys to (ideally) unique places in memory. This makes it possible to “look up” the values efficiently, even though the items are not stored in sequential order.
- Due to efficiency concerns, most hash functions do not map keys uniquely, but use heuristics (compromises) to get around this problem.

Dictionary Functions

- Functions for displaying contents of dictionary:
  - dictionary.items(), dictionary.keys(), dictionary.values()
- Functions for changing dictionary:
  - dictionary.clear() – erases
  - my_dictionary.update([['bear','common_noun'], ['friendly','adjective']]) – argument can be sequence or dictionary – all pairs are added
  - my_dictionary.pop('book') removes item/returns value
  - my_dictionary.popitem() removes/returns arbitrary key/value pair
- Copying: my_dictionary.copy()
- Creating new dictionary with default value: {}.fromkeys(['see','do'],'verb')

Class Exercise (15 minutes)

1. Set a global variable called telephone_book to an empty dictionary

   telephone_book = {}

2. Write a simple function for users to add phone numbers into telephone_book. Remember to add *key+value* to *dictionary*, the command is:

   dictionary[key] = value

   The keys should be names (strings) and the values should be strings consisting of numbers and hyphens, e.g., '212-123-4567'

Class Exercise Part II

3. Add several names (at least 5) and phone numbers to your dictionary using the function you wrote.
4. Then write a new function called: create_inverse_phone_book
   - This function takes a dictionary as an argument and produces a new dictionary in which the keywords of the old dictionary are values and the values of the old dictionary are key words.
   - Hint: even though a dictionary is not a sequence, it can be treated as one for the purpose of a *for* loop.
5. Create a new global variable backwards_phonebook using this function.
6. Choose a way to the handle the case where 2 people share a phone number? Implement it if there's time.

Sets

- Sets are collections of 0 or more items with no duplicates
- Sets are unordered
- Sets have a similar syntax to dictionaries:
  - types_of_dog = {'dalmatian','setter','hound','pointer'}
- Sequences can be coerced into sets
  - types_of_dog = set(['dalmatian','setter','hound','pointer'])
- Normal sets are not used frequently
  - You can look them up dir(set) and help(set)
- Python dictionaries look sort of like sets of key value pairs.
  - However the behavior of functions (methods) and operators (like *in*) demonstrate some differences.

Summary 1

- We discussed 4 types of data structures:
  - Lists, tuples, dictionaries and sets
  - But we really focused mostly on lists and (less so) dictionaries
- Lists and dictionaries are both mutable
  - It is possible to change them after they are created
  - This has ramifications:
    - functions can change lists/dictionaries that are passed to them
    - lists/dictionaries can be self-referential

Summary 2

- Lists and tuples are sequences and have many of the properties that strings and ranges have, e.g., their behavior with indices, slices, for loops, etc.
- Dictionaries provide a way of looking up items without reference to sequences
  - Even though some dictionaries could be ordered (words in alphabetic order), this would be inconvenient/inefficient for very large sequences
  - Other dictionary keys have no natural order
- Random Numbers
  - Can be used to generate “random” art
  - The programmer/artist chooses which elements are random

Homework Reading

- Read Chapter 7

Homework: Create program that replaces holidays with dates

1. Create a dictionary of 8 or more holidays for 2011 listing dates as strings of the form '12/25/2011' (e.g., Christmas)
2. Write a function that uses the dictionary to replace holidays in a string with the corresponding dates. The function should work as follows:
   (a) It should splits the string into words (use the split function)
   (b) It should check if each word is in the dictionary and replace it in the list with its value, e.g., the list ['I','wish','you','a','merry','Christmas'] should become: ['I','wish','you','a','merry', '12/25/2011']
   (c) Convert this list into a string using a for loop. Start with the empty string and add the items in the list, separated by spaces. Return the resulting string.
   (d) Your function in (b) should deal with punctuation and capitalization in some way. For capitalization, you could simply convert to lower case and only store lowercase items in your dictionary. For punctuation, you could strip off characters that are not letters or numbers.. Smoother ways of handling these issues are worth more points.

Extra Credit Assignments

- Please only do at most 1 extra credit assignment
- Extra Credit Assignments are optional
  - They can help boost your grade slightly (by raising the homework grade)
  - They cannot hurt your grade
  - You can get an A without doing the extra credit
- Grading Criteria: does the program work?; Is it clever?; Is it amusing/interesting?; Is the code clear?
- It might be fun (if you like this sort of thing)
- Extra Credit is due the 2nd to last class.

There are 2 possible Extra Credit Assignments

- Both involve random numbers and sequences
- Both involve modifying existing programs
- One involves extending the random turtle drawing project.
- One involves extending the random sentence creation program.

Extra Credit Assignment A

1. Modify the Turtle program written in class (slides 14 to 16)
   - Randomly generate similar lists consisting of:
     - combinations of 'square', 'triangle', 'circle' and 'left'
     - And random numbers from 1 to 10 for physical dimensions and 10 to 90 for angles
   - Have the program use 'split' to divide the numbers from the commands so that the turtles can have more variety in their random creations
2. Add some fun stuff. These are examples, but do what you want (don't go too wild):
   - Variety of Shapes, Colors, etc.
   - Weights, so some actions are more likely than others
   - You could record “addresses” in a dictionary (address_book) indicating where certain shapes are. The x,y coordinates of each turtle position is returned by turtle.pos(). You could then have a separate function which causes the turtle to go to the location of a particular shape by looking it up in the address_book, e.g., go_to_shape(blue_triangle) would cause the turtle to go to the last blue triangle that it drew..
   - Incorporate your stick figure functions from the previous turtle assignment

Extra Credit Assignment B – Slide 1

- Modify the random sentence program to either make it work better or to tune it to a particular purpose
- Possibility 1: fix the grammar in small ways. Examples:
  - If 'a' comes before a single noun that begins with a vowel, use 'an' instead.
  - Intransitive verbs (laugh, rose, etc.) should occur without a following noun group. Change the program to allow for this (make a separate word list)
- Possibility 2: Complicate the grammar, e.g., add a Prepositional Phrase:
  - A preposition (of, in, on, about, for, to, …) + a Noun Group
  - Can occur: after the noun in a noun group or at the end of a verb group

Extra Credit Assignment B – Slide 2

- Possibility 3: Adapt the Program for Some Purpose
  - Add words associated with a particular domain, e.g., the subject (first noun group) could optionally be taken from a random list of names of famous people.
  - Add verbs/adjectives from a particular semantic domain: biology, sports, food, romance
- Etc..