6.00 Recitation 4
28 February 2014

Administrivia
● pset 3 is due on Tuesday (March 4)
● quiz 1 is on Thursday (March 6)
   ○ email Emily (emilyrsu@mit.edu) by tonight if you have a conflict
   ○ review session 7:30-9:30pm Monday (March 3)
   ○ 7-9pm OH on Monday (March 3) cancelled due to review session
   ○ lecture on Thursday (March 6) cancelled, optional OH instead
   ○ closed book, can have 2 double-sided sheets of notes
● code for this recitation is in recitation4.zip

Debugging and Testing

Debugging

Printing out the values of your variables is incredibly useful. 
Google is your friend if you encounter an error you don’t understand.

Using assertions:
   assert <boolean condition>
   assert <boolean condition>, <argument>

Testing
● Glass-box testing: explore paths through code
● Black-box testing: explore paths through specification
● Unit testing: Test each function that you write to make sure it works before testing composite behavior.
● Regression testing: Keep tests around. Make change to the code. Retest with older tests to make sure you haven’t broken anything.
● Test first programming: Write test method before writing a single line of code.

Exercises
Try fixing the bugs in rec4_buggy.py. (rec4_correct.py is the version with the bugs fixed.)

Exceptions (and how to handle them)
Exceptions occur when the syntax is correct but the code performs some operation that isn’t allowed (e.g. saying int(‘1.1’) or trying to divide by zero).

References
● http://docs.python.org/2/tutorial/errors.html
• Sections 7.1 and 7.2 in the textbook
• Python has a ton of built-in exceptions. The whole list of them is here: http://docs.python.org/2/library/exceptions.html

Terminology
• you raise (or throw) an exception when you want an exception to occur: raise
• you handle (or catch) an exception when you want to do something (and not have the program crash) in the case you encounter an exception: try/except
• an unhandled exception will cause a Traceback (or stack trace) to be printed to the interactive shell (in IDLE it’s printed in red)

Exception Handling

try/except
In the following example, if a ValueError (or a subclass of ValueError) is raised (i.e. happens) within the try clause, then whatever is in the try clause after the erroring line is not executed, and the program jumps to the except clause:
try:
...
except ValueError:
...

If you say “except” without specifying a specific exception, then it handles ALL exceptions that occur in the try block:

try:
...
except:
...

raise
You can raise a ValueError in your code by saying:
raise ValueError

Within the except clause, saying “raise” will re-raise the exception.
try:
...
except ValueError:
...
raise

Examples (What does this print?)
Code in rec4.py.
[Python will raise an exception if you say int(‘1.1’) but not if you say int(‘1’)]
• multiply_1: int('1') doesn’t cause an error, so the except clause isn’t entered
• multiply_2: what happens before and after the try/except occurs normally, but we enter the except clause because int(‘1.1’) causes an error
• multiply_3: the print statement after the except clause is not executed because we re-raise the exception within the except clause

Tuples and Dictionaries

Tuples
• Are like lists, but immutable
• Why are they useful?
  ○ As keys in dictionaries (lists can’t because they are are mutable)
  ○ Swapping two variables (x, y = y, x)
  ○ Returning multiple values from a function

Dictionaries
• Key, value pairs
• Keys can be integers, strings, tuples, etc. (anything immutable)
• Keys can’t be lists, dictionaries, etc. (anything mutable)
• Keys are unique, values don’t have to be

rec4.py has an example that shows useful dictionary functions.

Python documentation for dictionary functions:
http://docs.python.org/2/library/stdtypes.html#mapping-types-dict

Dictionary lookup is fast, compared to if you were to store key/value pairs by using lists.
rec4_names.py is a demo that maps name/gender pairs showing how much faster dictionary lookup is compared to list lookup. This code requires names.txt.

(I got the names from the 2012 data for “National Data on the relative frequency of given names in the population of U.S. births where the individual has a Social Security Number” and processed the data such that any name that appears for both male and female becomes unknown - apparently there are either administrative errors or some people who name their baby boys “Emily”...)

Mutations and Aliasing

Important Concepts
• Mutation: modification of an object
• Immutable types: list, dictionary
• Immutable types: tuple, string (also float, integer, boolean)
• Aliasing: two variables bound to the same object (i.e. two different names for one thing)
• Side effect: when a function mutates variables
Sorting and Reversing Lists

Given a list L:
- L.sort() sorts the list in place (mutates L)
- sorted(L) returns a new list that is sorted (doesn’t mutate L)
- L.reverse() reverses the list in place (mutates L)
- reversed(L) doesn’t return a list but can be used like “for elem in reversed(L):” [it returns a generator, which is mostly outside the scope of this class]

Exercises (What does this print?)
See rec4.py.