2016-04-06

William A. Stein

4/6/2016

Contents

1 Math 480: Open Source Mathematical Software

1.0.1 2016-04-06

1.0.2 William Stein

1.1 Lectures 5: Basic Python Data Structures: lists, tuples, dicts

Misc:

1. The TA has the flu, so his office hours tomorrow probably canceled (or unsafe).
2. Homework due Friday at 6pm. I've almost finished implementing peer grading :-)
3. Start the screencast!!!!

1.2 List Comprehensions

A clean way to make a list of things. You might then iterative over this list, or do something else with it. Very handy in mathematics.

[expression(x) for x in [list]]
[expression(x) for x in [list] if something(x)]

[expression(x, y) for x in [list] for y in [another list] if something(x, y)]
etc.

```
[n for n in range(-3,5)]  # you can give a start value for range
```

```
[n*2 for n in range(10)]
```

```
[n for n in range(20) if n%2 == 0]
```

```
[[x,y] for x in range(4) for y in ['X','Y','Z'] if x%2 == 0]
```

You can typically rewrite a list comprehension as a for loop (or nested for loops).

```
v = []
for n in range(10):
    v.append(n)
print v
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

KEY INSIGHT: The order of the for loops and the if are exactly the same as in the list comprehension.

```
v = []
for x in range(4):
    for y in ['X','Y','Z']:
        if x % 2 == 0:
            v.append([x,y])
print v
```

```
[[0, 'X'], [0, 'Y'], [0, 'Z'], [2, 'X'], [2, 'Y'], [2, 'Z']]
```

Exercise now:
Construct the following list using a list comprehension and the if/else exercise function my_sign you wrote above:
[[-3,'negative'], [-2,'negative'], [-1,'negative'], [0,'zero'], [1,'positive'], [2,'positive'], [3,'positive']]
If you have time, do the same but using a for loop.

1.3 Python Lists

WARNING: Python lists are SUBTLE due to references.

A Python list is a an ordered sequence of references to Python objects. And assignment in Python is copy by reference.

This is totally different than how things work with, e.g., MATLAB.

Consider first this:

```
a = [1,2,3]
b = a
b[0] = 5
```

```
print "a = ", a
print "b = ", b
```

And here is what happens in MATLAB, actuallyOctave is a free clone of matlab (1-based and copy by value):

```
%octave
a = [1,2,3];
b = a;
b(1) = 5;
a
b
```

And R (1-based and copy by value):

```
%r
a <- c(1,2,3);
b <- a;
b[1] = 5;
a
b
```

```
[1] 1 2 3
[1] 5 2 3
```

Here's what happens in Javascript (0-based and copy by reference):

```
%javascript
a = [1,2,3];
b = a;
b[0] = 5;
print("a =",a)
print("b =",b)
```

And in Julia (1-based and copy by reference!):

```
%julia
a = [1,2,3];
b = a;
b[1] = 5;
println("a =", a, "\nb =", b);
```

```
a =[5,2,3]
b =[5,2,3]
```

The point is that different programming languages make very subtle and different design choices. And that's all they are choices. One choices really is better for some purposes and worse for other purposes.

The copy by reference choice of Python makes lists potentially subtle. For example:

```
v = [1,2,3]
w = [v, v, [1,2,3]]
w
```

```
w[0][0] = 'sage'
```

Now what the heck is w equal to? (Wait until lots of people in class think this through.)

```
print w
```

Moral: Just because something prints out at [[1, 2, 3], [1, 2, 3], [1, 2, 3]] doesn't mean you know what that something is! In Python, things can be very subtle.

Don't judge an object by its cover (how it prints).

If you don't get this point, you might hate Python. If you do, you might start to see huge power and flexibility in what Python offers and understand why Python might be the most popular programming language in data science, which is itself a very popular emerging field.

Shallow and deep copies.

Let's try the copy module from Python's standard library: https://docs.python.org/2/library/

```
import copy  # this is how you import a module in Python
v = [1,2,3]
w = [v, v, [1,2,3]]
w1 = copy.copy(w)   # how to use a function defined in a module
w1[0][0] = 'hi'
print "w1 = ", w1
print "w  = ", w    # not much of a copy!!!  It just copies the top \
   level references

# Question: will this change w?
w1[0] = 'hi'
print w   # what do you expect
```

```
# To really copy, use deepcopy:
import copy  # this is how you import a module in Python
v = [1,2,3]
w = [v, v, [1,2,3]]
w1 = copy.deepcopy(w)   # uses more memory; causes more work -- and \
   that's why we want options
w1[0][0] = 'hi'
print "w1 = ", w1
print "w  = ", w    # yep, not touched by changing w
```

MORAL: Python is a real programming language designed by software engineers. It's not a special-purpose math-only language, and it respects deeper, and more powerful, subtle (and possibly confusing) ideas from programming.

1.4 Some more about data structures: tuples, dicts, sets

1.4.1 tuples

- Like Python lists, except you use parens instead of square brackets
- You can't change the number of entries or what they reference. However, you might be able to change the referenced thing itself.

```
v = [['x','y'], 2, 5]  # a list
t = (['x', 'y'], 2, 5) # a tuple
print "v =", v
print "t =", t
```

```
v[1] = 3
```

```
v.append("something")   # lists allow for lots of exciting ways of \
   changing them
v
```

```
del v[3]
v
```

```
t[1] = 3  # not allowed!
```

```
t.append  # not so for tuples
```

```
# However, and this is critical -- you can change v[0] or t[0] \
   itself!

v[0].append('z')
v
```

```
t[0].append('z')   # this doesn't change t --
# think of t as references to 3 specific objects -- you are changing\
    one of the referenced objects, not t itself!
t
```

Exercise right now: Write a function that takes as input a tuple and returns the sorted version of the tuple.

```
# Hints to make this really easy:
v = (1,7, 4)
sorted(v)   # returns sorted list obtained from a tuple (or whatever\
   )
```

```
tuple([4,7,8])  # tuple of a list makes a tuple
```

```
def sorted_tuple(x):
    return  # what goes here? something

sorted_tuple((7,3,5))  # test your function!
```

1.5 Another important data structure: the Python dictionary

A Python dictionary is the Python version of what computer scientists might call an "associative array".

It's what us mathematicians call a "function" (from one finite set to another). Except, like a list, it can be modified (to make another function).

```
d = {5:25, 10:100, 3:8}

d

d[5]

d[10]

d[3]

d[7]  # should fail -- since we didn't say what 7 maps to.
```

The keys (inputs) of a Python dictionary can be any object x in Python where hash(x) doesn't give an error.

In practice, hash(x) is supposeed to work if and only if x is immutable, i.e., really can't change.

```
hash(7)

w = [2,3]
hash(w)   # better not work!

w = (2,3)
hash(w)    # a tuple where each thing in the tuple is immutable

d = {(2,3):5,   (7,19): 26}   # you can use anything immutable as \
   the inputs (or keys or domain)
d[(7,19)]

w = ([2,3], 5)   # a tuple where first thing is NOT immutable
hash(w)
```

Exercise: Which of the following are immutable, i.e., hash-able, so they can be used as keys for a dictionary?

- the string "Foo"

- the empty list []
- the number 3.14
- the dictionary 7:10, 3:8
- the tuple ('foo', bar')

Make a dictionary that has all the immutable objects above as keys and anything you want (hashable or not) as values.