Quiz #1: Review of Iteration and Python Data Types ICS-33 Fall 2015

When working on this quiz, recall the rules stated on the Academic Integrity statement that you signed. You can download the `q1helper` project folder (available for Friday, on the **Weekly Schedule** link) in which to write/test/debug your code. Submit your completed `q1solution.py` module online by Thursday, 11:30pm. I will post my solutions to EEE reachable via the **Solutions** link on Friday morning.

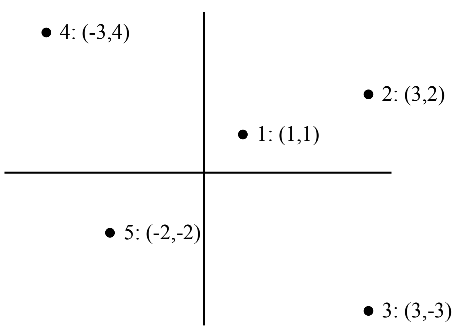
---

1. (5 pts) Define the following two functions: each returns a function (that can be called) as its result.

(a) The `compose` function takes two univariate function arguments; it returns a function that takes a single argument and returns the result of calling first function on the result of calling the second function on the argument. For example: if we define the name `f = compose(lambda x : 2*x, lambda x : x+1)` then `f(5)` returns `12` (it first adds `1` to `5` and then doubles that result.

(b) The `self_compose` function takes one univariate function and one `int` argument (raise `AssertionError` if the `int` is `< 0`). Call the parameters `f` and `n` respectively; it returns a function that takes an `int` as an argument and returns `f(f(f(...f(x)...)))` where `f` is called `n` times on its argument `x`: For example: if we define `sc = self_compose(lambda x : 2*x),5)` then `sc(1)` returns `32`. Note that if n is 0, `f(x)` returns `x`.

2. (5 pts) Define the following three sorting functions: each returns a list of values that has been sorted in a different way. Your function bodies must be exactly one line long: just a `return` statement. The argument to each function is a dictionary in the form `{int:(int,int)}` showing the ordinal of a point (1st point, 2nd point. 3rd point, etc.) as a key, and a `2-tuple` representing the key's x,y coordinate in the form `(x,y)`. For example, `ps = {1:(1,1),2:(3,2),3:(3,-3),4:(-3,-4),5:(-2,-2)}` means that the 1st point is at coordinate (1,1); the 2nd point is at coordinate (3,2); the 3rd point is at coordinate (3,-3); etc. Examine the picture below, showing these five points.



(a) The `sorted1` function returns a list of `2-tuples` (ordinals and their points), sorted ascending by x coordinate: points with equal x coordinates should be sorted descending by y coordinate: for the `ps` dictionary above the result is `[(4,(-3,4)),(5,(-2,-2)),(1,(1,1)),(2,(3,2)),(3,(3,-3))]`

(b) The `sorted2` function returns a list of `2-tuples` (not the ordinals, just the points), sorted ascending by the angle each point forms when a line is drawn to it from the origin: for the `ps` dictionary above the result is `[(-2,-2),(3,-3),(3,2),(1,1),(-3,4)]`. Hint: from the `math` module, we can call `atan2(y,x)` to compute the angle corresponding to the point (x,y). The angles returned from `atan2` vary from $-\pi$ (West in 3rd quadrant), through $-\pi/2$ (South, bordering 3rd-4th quadrant), through 0 (East, bordering 4th-1st quadrant), through $\pi/2$ (North, bordering 1st-2nd quadrant), through π (West in 2nd quadrant). So, ascending angles go from West in quadrant 3, counter-clockwise to West in quadrant 2.

(c) The `sorted3` function returns a list of `ints` (just the ordinals, not the points), sorted ascending by the angle each point forms when a line is drawn to it from the origin (same criteria as above, but returning ordinals not points): for the `ps` dictionary above the result is `[5,3,2,1,4]`.

3. (5 pts) Define the following two functions using comprehensions to compute their results. Your function bodies must be exactly one line long: just a **`return`** statement. Both functions take the same kinds of parameter as the sorted functions in Problem 2.

(a) The **`points`** function returns a list of **`2-tuples`** (not the ordinals, just their points), sorted ascending by their ordinal values: for the **`ps`** dictionary above the result should be **`[(1,1),(3,2),(3,-3),(-3,4),(-2,-2)]`**.

(b) The **`first_quad`** function returns a dictionary whose keys are **`2-tuples`** (points) that are in the first quadrant (points whose x and y coordinates not negative), whose associated values are their distance from the origin: for the **`ps`** dictionary above the result is **`{(3,2):3.605551275463989,(1,1):1.4142135623730951}`**. Hint: import the **`sqrt`** function from the **`math`** module to compute the distance.

The next two problems (4 and 5) use a dictionary that is stores a database of phone call meta-data (just how many times **callers** call **callees**). The form of this database is a **`dict`** associating a **`str`** (the **caller**) with another **`dict`** of all the people they called: this inner **`dict`** associates a **`str`** (the **callee**) with an **`int`** (the number of times the **caller** called the **callee**). For example, a simple/small database of **`a`** calling **`b`** **`2`** times, **`a`** calling **`c`** **`1`** time, **`b`** calling **`a`** **`3`** times, **`b`** calling **`c`** **`1`** time, **`c`** calling a **`1`** time, and **`c`** calling **`d`** **`2`** times would be stored as **`db = 'a':{'b':2,'c':1},'b':{'a':3,'c':1},'c':{'a':1,'d':2}}`**.

4. (5 pts) Define the following functions.

(a) Define the **`called`** function, which returns a dictionary associating a **`str`** (the **caller**) with an **`int`** (the number of times the **caller** called anyone). **`called(db)`**, returns **`{'a':3,'b':4,'c':3}`**. Your function body must be exactly one line long: just a **`return`** statement using a comprehension to create a **`dict`**.

(b) Define the **`got_called`** function, which returns a dictionary associating a **`str`** (the **called**) with an **`int`** (the number of times any called **called**). **`got_called(db)`** returns a dictionary value equivalent to **`{'a':4,'b':2,'c':2,'c':2}`**. Hint: use a **`defaultdict`** for the dictionary, for the simplest code.

5. (5 pts) Define the **`invert`** function, which returns a dictionary associating a **`str`** (the **callee**) with a **`dict`** of all the people they were called by: this inner **`dict`** associates a **`str`** (the **caller**) with an **`int`** (the number of times the **callee** was called by the **caller**). **`invert(db2)`** returns the dictionary value equivalent to **`{'a':{'b':3,'c':1},'b':{'a':2},'c':{'a':1,'b':1},'d':{'c':2}}`**. Hint: use a **`defaultdict`** for the outer dictionary, for the simplest code.