Programming, Probability, and the Modern Mathematics Classroom Exercises — Part 4

Manan Shah
Mathematician-At-Large

June 5, 2013

Please make sure to have read the blog post with the same topic name on the Math Misery website, otherwise this will be out of context.

Some reminders

- Have students to comment the code, line by line.
- Let students tweak, modify, and otherwise alter the code.
- Don't worry about programming formalities.
- Have fun, relax, and learn.

The Python set object

Here I'll show how the Python `set` is used and give a small example of it in operation via the function `timeRequiredToCollectItems`. There is far more that can be done beyond what is given here and as usual I encourage instructors to let students explore. A few things to note in the command line write-up below. I use `random.choice`, something not previously discussed, but now has snuck in.

```
randomobject = random.choice(listOfObjects)
```

In previous exercises, I would get the length of the list, choice an index at random and then get the specific object by referencing the list at that index. Python is a very nice language with lots of built-in functionality. `random.choice` is one of many such niceties. If this wasn't already discovered, you now know of it.

I use the `while` statement as well, something that wasn't previously discussed. Again, given Python's intuitive syntax and readability, what the `while` block does should be straightforward.

Finally, there is this line of code using the dictionary object:

```
drawsCounterDictionary[x] = drawsCounterDictionary.get(x,0) + 1
```

This is a nice simple way of saying "If x is not in the dictionary then assign to the key x the default value of 0, then add 1."

The example used is the "Coupon Collector Problem." This is a nice probabilistic problem to introduce to students because it will help them to understand:

- that things can be streaky — an object with four items in it took 42 draws to collect!
- that dictionaries are nice as counters for unknown events
- that sets, dictionaries, lists can all work together.

```
>>> pythonset = set()
>>> ## sets are different from dictionaries
>>> ## sets collect unique objects and can’t be indexed
>>> ## we will explore how some of the properties of sets in Python
>>> type(pythonset)
<class ’set’>
>>> len(pythonset)
0
>>> pythonset.add(233)
>>> len(pythonset)
1
>>> pythonset
{233}
>>> pythonset[0]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    pythonset[0]
TypeError: ’set’ object does not support indexing
>>> pythonset.remove(233)
>>> pythonset
set()
>>> pythonset.add(432)
>>> pythonset
{432}
>>> pythonset.add(432)
>>> pythonset
{432}
>>> ## notice that once an element has been added to a set, it can’t be added again
>>> ## compare this against Python lists
>>> pythonlist = []
>>> pythonlist.append(432)
>>> pythonlist
[432]
>>> pythonlist.append(432)
>>> pythonlist
[432, 432]
>>> ## why use sets?
>>> ## maybe we are listening for events
>>> ## and once we have received N events, we’ll do something else
>>> ## consider the classical coupon collector problem
>>> ## suppose we are drawing integers in [1,10] at random with replacement
>>> ## we want to know how long, on average, it will take for us to collect
>>> ## all ten integers
>>> ## sets are useful here because the lookup to check if an element exists
>>> ## is very fast, unlike that in lists where one would have to traverse
>>> ## the list to see if the element exists
>>> def timeRequiredToCollectItems(listOfObjects):
    numobjects = len(listOfObjects)
    numberofdraws = 0
    itemsCollectedSet = set()
    while len(itemsCollectedSet) != numobjects:
        randomobject = random.choice(listOfObjects)
        itemsCollectedSet.add(randomobject)
        numberofdraws += 1
    return numberofdraws

>>> myobjects = ["Jack", "Jill", 343, ("hello","this is a tuple in a list")]
>>> import random
>>> timeRequiredToCollectItems(myobjects)
6
>>> timeRequiredToCollectItems(myobjects)
6
>>> timeRequiredToCollectItems(myobjects)
5
>>> ## now we'll use a dictionary to keep track of (draws, occurrences)
>>> drawsCounterDictionary = {}
>>> for i in range(10000):
    x = timeRequiredToCollectItems(myobjects)
    drawsCounterDictionary[x] = drawsCounterDictionary.get(x,0) + 1


>>> for drawtime in sorted(drawsCounterDictionary):
    print(str(drawtime) + " draws were required ", end = '')
    print(str(drawsCounterDictionary[drawtime]) + " times out of 10000.")


4 draws were required 927 times out of 10000.
5 draws were required 1418 times out of 10000.
6 draws were required 1466 times out of 10000.
7 draws were required 1367 times out of 10000.
8 draws were required 1107 times out of 10000.
9 draws were required 903 times out of 10000.
10 draws were required 636 times out of 10000.
11 draws were required 519 times out of 10000.
12 draws were required 384 times out of 10000.
13 draws were required 301 times out of 10000.
14 draws were required 255 times out of 10000.
15 draws were required 195 times out of 10000.
16 draws were required 111 times out of 10000.
17 draws were required 111 times out of 10000.
18 draws were required 68 times out of 10000.
19 draws were required 58 times out of 10000.
20 draws were required 44 times out of 10000.
21 draws were required 34 times out of 10000.
22 draws were required 30 times out of 10000.
23 draws were required 14 times out of 10000.
24 draws were required 11 times out of 10000.
```

```
25 draws were required 8 times out of 10000.
26 draws were required 11 times out of 10000.
27 draws were required 8 times out of 10000.
28 draws were required 4 times out of 10000.
29 draws were required 2 times out of 10000.
30 draws were required 1 times out of 10000.
31 draws were required 2 times out of 10000.
32 draws were required 1 times out of 10000.
33 draws were required 2 times out of 10000.
35 draws were required 1 times out of 10000.
42 draws were required 1 times out of 10000.
```

Summary

In this part, we went over the Python `set` object by using the coupon collector problem. We also suggest that instructors let students explore the other methods that are available on `set` objects — like using `pop` to remove items from the set, `intersection`, `difference`, etc. to perform set-like operations. We discuss this in the next set of exercises.

If you need help, have questions, or would like to set up a workshop at your school get in touch with me at help@mathmisery.com.