homework-5

April 24, 2016

1 Introduction to Python

2 Homework #5

3 Due Friday April 29 , 1:50pm

4 Academic Honesty

- The computer science department has strict polices. Check the department web page for details.
- Do not look at anybody else's source code. Do not show anybody your source, or leave your source where somebody could see it. You MUST write your own code.
- For this class, feel free to discuss issues with other people, but suggest waiting an hour or two after a discussion, before writing your code.
- Cases of non original source will be refered to the Judical Committee.

5 Reminder

- if you did the first four homeworks and are happy with them, you do NOT need to do this one

6 Problem 1

- Python is very popular in finance
- use pandas data frames to analyse stocks
- grab data from Yahoo

```
In [52]: import pandas as pd

         def getStockDF(symbol):
             return pd.read_csv("http://chart.yahoo.com/table.csv?s=" + symbol)

         apple=getStockDF('AAPL')
```

```
In [27]: # quick summary of the data
         # these are NOT the results you will get below, because we are using
         # a subset of this data

         apple.describe()
```

```
Out[27]:              Open         High          Low        Close        Volume  \
         count  8917.000000  8917.000000  8917.000000  8917.000000  8.917000e+03
         mean     99.301647   100.545493    97.945121    99.256306  9.172433e+07
```

```
std     138.376940   139.534988   136.997716   138.290790  8.826014e+07
min      11.125000    11.125000    11.000000    11.000000  2.504000e+05
25%      26.312500    26.750000    25.875000    26.250000  3.703280e+07
50%      42.249999    42.880000    41.374999    42.125000  6.412000e+07
75%      96.950003    98.779998    95.169998    96.999997  1.142155e+08
max     702.409988   705.070023   699.569977   702.100021  1.855410e+09

         Adj Close
count  8917.000000
mean     15.823733
std      29.771599
min       0.166046
25%       0.911696
50%       1.415725
75%      12.261422
max     130.671320
```

7 define 'stkstats'

- use trading data from the first trading day in 2005, to the last trading day in 2015
  - you can find these days from the data frame
- use 'Close' field for price calculations
- return a list of [close mean, close min, close max, close std, volume mean]

```
In [54]: # apple

         stkstats('AAPL')

Out[54]: [238.75427579776064,
          34.130000000000003,
          702.10002099999997,
          180.29574434543773,
          153458928.2051282]

In [55]: # google

         stkstats('GOOG')

Out[55]: [551.1103599263273,
          174.99030400000001,
          1220.1720359999999,
          186.48351707455066,
          8699994.980137235]
```

8 Problem 2

- You are in a store, and you have some cash burning a hole in your pocket - you want to spend all of it!!
- write 'pickitems'
- 1st arg - list of prices for things in the store. you can buy at most one item at each listed price
- 2nd arg - cash you have
- returns - list of prices that will exactly spend(sum up to)your cash
- itertools module is your friend

  - use brute force - try all combinations
  - there may be more than one solution - return the first one you find
- if you can't spend all your money, return None

```
In [1]: cash1 = 4
        prices1= [1,1,1,1,8]

        cash2 = 200
        prices2 = [150, 24, 79, 50, 88, 345, 3]

        cash3 = 8
        prices3 = [2, 1, 9, 4, 4, 56, 90, 3]

        cash4 = 542
        prices4 = [230, 863, 916, 585, 981, 404, 316, 785,
               88, 12, 70, 435, 384, 778, 887, 755, 740,
               337, 86, 92, 325, 422, 815, 650, 920, 125,
               277, 336, 221, 847, 168, 23, 677, 61, 400,
               136, 874, 363, 394, 199, 863, 997, 794, 587,
               124, 321, 212, 957, 764, 173, 314, 422, 927,
               783, 930, 282, 306, 506, 44, 926, 691, 568,
               68, 730, 933, 737, 531, 180, 414, 751, 28,
               546, 60, 371, 493, 370, 527, 387, 43, 541,
               13, 457, 328, 227, 652, 365, 430, 803, 59,
               858, 538, 427, 583, 368, 375, 173, 809, 896,
               370, 789]
```

```
In [3]: [pickitems(prices1, cash1), pickitems(prices2, cash2),
         pickitems(prices3, cash3), pickitems(prices4, cash4)]
```

```
Out[3]: [(1, 1, 1, 1), (150, 50), (4, 4), (221, 321)]
```

9 Problem 3

- define a function decorator 'secure'
- secure adds two required arguments before any others, a 'user' and a 'password'
- if the user is not registered, raise an Exception
- if the user is registered, but the password is wrong, raise an Exception

```
In [19]: # just use a dict as the user/password 'database'

         up = dict()
         up['jack'] = 'jackpw'
         up['jill'] = 'jillpw'


         @secure
         def foo(a,b):
             return (a+b)

         @secure
         def bar(a, b=34):
             return(a+b)
```

```
In [20]: # bad call - no user/pw

         foo(1,2)


         ---------------------------------------------------------------------------

         Exception                                 Traceback (most recent call last)

         <ipython-input-20-750e3ba2ca5c> in <module>()
           1 # bad call - no user/pw
           2
     ----> 3 foo(1,2)


         <ipython-input-18-6e65dc21854f> in __call__(self, user, pw, *pos, **kw)
           6
           7         if not user in up:
     ----> 8             raise Exception('User {} not registered'.format(user))
           9         if pw != up[user]:
          10             raise Exception('Bad password {} for user {}'


         Exception: User 1 not registered


In [21]: # good call

         foo('jack', 'jackpw', 1 ,2)

Out[21]: 3

In [22]: # bad user

         foo('frank', 'bad', 1 ,2)


         ---------------------------------------------------------------------------

         Exception                                 Traceback (most recent call last)

         <ipython-input-22-b35ba770f676> in <module>()
           1 # bad user
           2
     ----> 3 foo('frank', 'bad', 1 ,2)


         <ipython-input-18-6e65dc21854f> in __call__(self, user, pw, *pos, **kw)
           6
           7         if not user in up:
     ----> 8             raise Exception('User {} not registered'.format(user))
           9         if pw != up[user]:
          10             raise Exception('Bad password {} for user {}'
```

```
Exception: User frank not registered
```

```
In [24]: # good user, bad passwd

         foo('jill', 'nope', 3, 4)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)

<ipython-input-24-ebb563ec7723> in <module>()
  1 # good user, bad passwd
  2
----> 3 foo('jill', 'nope', 3, 4)


<ipython-input-18-6e65dc21854f> in __call__(self, user, pw, *pos, **kw)
  9         if pw != up[user]:
 10             raise Exception('Bad password {} for user {}'
---> 11                             .format( pw, user))
 12         return(self.f(*pos, **kw))


Exception: Bad password nope for user jill
```

```
In [25]: # works with keywords

         bar('jill', 'jillpw', 5, b=34)
```

```
Out[25]: 39
```

10 Problem 4

- write short helper functions for problem 5

11 write isPrime predicate

- first check some special cases, then starting with 3, try each odd number until the square of the odd is bigger than the number being tested
- use '%' to check divisors
  - for example, 0 == n % 2 will be true if n is even

```
In [3]: [[n,isPrime(n)] for n in [1,2,3,8,23,25,31,33,91]]
```

```
Out[3]: [[1, False],
         [2, True],
         [3, True],
         [8, False],
         [23, True],
         [25, False],
         [31, True],
         [33, False],
         [91, False]]
```

12 write genPrimes

- a prime number generator

```
In [45]: import itertools

         g = genPrimes()

         list(itertools.islice(g, 0, 20))
```

```
Out[45]: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]
```

13 write differences

- computes the difference between adjacent elements
- result list is one shorter than input list

```
In [12]: differences([1,3,4,22,33])
```

```
Out[12]: [2, 1, 18, 11]
```

14 define same

- true if all list elements are the same

```
In [14]: [same([]), same([3,3,3,3,3]), same([3,3,3,3,3,8,3])]
```

```
Out[14]: [True, True, False]
```

15 define arithmetic

- in an arithmetic sequence, the difference between consecutive elements is the same
- use differences and same

```
In [17]: [arithemetic([0,2,4]), arithemetic([3,7,9])]
```

```
Out[17]: [True, False]
```

16 define primes4

- find all the 4 digit primes

```
In [46]: p4 = primes4()
         [len(p4),p4[:10], p4[-10:]]
```

```
Out[46]: [1061,
          [1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061],
          [9887, 9901, 9907, 9923, 9929, 9931, 9941, 9949, 9967, 9973]]
```

17 Problem 5

- look at the 4 digit primes
- it turns out there are two triples of primes such that:
  - the digits of the primes are permutations of each other
  - the primes in increasing order are an arithmetic sequence
- one of the triples is [2969, 6299, 9629]

```
In [33]: trip=[2969, 6299, 9629]
         [differences(trip),arithemetic(trip), list(map(isPrime, trip))]
```

```
Out[33]: [[3330, 3330], True, [True, True, True]]
```

18 write findTriples

- findTriples has three parts:
  - invent a scheme for generating a key from the primes
    - for example, 1013, 1031, 1103, 1301, 3011 should all map into the same key
  - build a dictionary
    - a key's value should hold all the 4 digit prime permutations of a given digit set
  - once the dictionary is built:
    - for each set of numbers in a key value,
    - try every length 3 combination and see if it is an arithmetic sequence
    - itertools.combinations is your friend - note it preserves sequence order
  - return a list of the triplets

```
In [63]: t1, t2 = findTriples()
         [t2, differences(t2)]
```

```
Out[63]: [(2969, 6299, 9629), [3330, 3330]]
```