homework-4

October 12, 2015

1 Introduction to Python

2 Homework #4

3 Due Tuesday Oct 20 , 11:50pm

4 Problem 1 - Longest Ordered Substring(los)

- argument: a string
- returns: the longest substring such that the characters strictly increase(in the $<$ sense) from left to right
- if there is more than one longest string, return any one of them

```
In [1]: # usual ordering
        ['a' < 'b', 'b' < 'a', 'a' < 'a']

Out[1]: [True, False, False]

In [4]: los('abcd')

Out[4]: 'abcd'

In [5]: los('xabcyz')

Out[5]: 'abcyz'

In [6]: los('cba')

Out[6]: 'c'

In [7]: los('larry')

Out[7]: 'ar'

In [8]: los('xabcxabcdexuvwxyz')

Out[8]: 'abcdex'

In [9]: los('xabcxabcdexpuvwxyz')

Out[9]: 'puvwxyz'
```

5 Problem 2

- suppose we want to convert between C(Celsius) and F(Fahrenheit), using the equation $9C = 5$(F-32)
- write functions 'c2f' and 'f2c'
- do all computation in floating point for this problem

```
In [4]: [c2f(0), c2f(100), f2c(32), f2c(212)]
```

```
Out[4]: [32.0, 212.0, 0.0, 100.0]
```

- to write f2c, you solved the equation for C, and made a function out of the other side of the equation
- to write c2f, you solved for F, ...
- there is another way to think about this
- rearrange the equation into a symmetric form

```
9*C - 5*F = -32*5
```

- you can think of the equation above as a "constraint" between F and C. if you specify one variable, the other's value is determined by the equation. in general, if we have

```
c0*x0 + c1*x1 + ... cN*xN = total
```

- cI are fixed coefficients
- specifying any N of the (N+1) x's will determine the remaining x variable
- define a class, 'Constaint' that will do 'constraint satisfaction'

```
In [60]: # setup constraint btw C and F
         # 1st arg is var names,
         # 2nd arg is coefficients
         # 3rd arg is total
         c = Constraint('C F', [9,-5], -5*32)

         # 1st arg - variable index or name
         # 2nd arg - variable value
         # setvar will fire when there is only one unset variable remaining
         # it will print the variable values, and returns them in a list, and
         # clear all variable values
         c.setvar(0, 100)
```

```
C = 100.000000
F = 212.000000
```

```
Out[60]: [100.0, 212.0]
```

```
In [52]: # can set var by index or name
         c.setvar('C', 0)
```

```
C = 0.000000
F = 32.000000
```

```
Out[52]: [0.0, 32.0]
```

```
In [63]: c.setvar('F', 212)
```

```
C = 100.000000
F = 212.000000
```

```
Out[63]: [100.0, 212.0]
```

```
In [54]: # more complex example
         c2 = Constraint('x0 x1 x2 x3 x4', range(5), 1)

In [55]: c2.setvar('x0', 0)

In [56]: c2.setvar('x1', 10)

In [57]: # x2
         c2.setvar(2,20)

In [58]: # only one unset var left, x4, so fire
         c2.setvar('x3', 30)

x0 = 0.000000
x1 = 10.000000
x2 = 20.000000
x3 = 30.000000
x4 = -34.750000

Out[58]: [0.0, 10.0, 20.0, 30.0, -34.75]
```

6 Problem 3

- write 'mindot'. given two equal length vectors, different dot products can be calculated, by permutting the order of the vectors. for example, given the vectors

  ```
  [10,20]
  [0, 1]
  ```

- there are two possible dot products, 10 and 20.
- mindot should return the min, 10
- mindot can be written very easily by using functions from itertools and functools
- 'operator' module has functional versions of operators
- '+' <=> operator.add
- '*' <=> operator.mul

```
In [29]: import itertools
         import functools
         import operator

         v2a = [10,20]
         v2b = [0, 1]

         v3a = [1,3,-5]
         v3b = [-2, 4, 1]

         v4a = range(1,6)
         v4b = [1,0,1,0,1]

         [operator.add(2,3), operator.mul(2,3)]

Out[29]: [5, 6]

In [62]: [mindot(v2a,v2b),mindot(v3a, v3b), mindot(v4a, v4b)]

Out[62]: [10, -25, 6]
```

7 Problem 4

- 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
- 2nd arg - cash you have
- returns - list of prices that will exactly spend your cash
- itertools module is your friend

```
In [7]: 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 [12]: [pickitems(prices1, cash1), pickitems(prices2, cash2), pickitems(prices3, cash3), pickitems(pr
```

```
Out[12]: [(1, 1, 1, 1), (150, 50), (4, 4), (221, 321)]
```

8 Problem 5

- 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 [2]: # the user/password 'database'
        up = {}
        up['jack'] = 'jackpw'
        up['jill'] = 'jillpw'


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

        @secure
```

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

In [3]: # wrong number of args

        foo(1,2)


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

        Exception                                 Traceback (most recent call last)

        <ipython-input-3-790da69007fa> in <module>()
          1 # wrong number of args
          2
    ----> 3 foo(1,2)


        <ipython-input-1-7281da65c0b8> in __call__(self, user, pw, *pos, **kw)
          6
          7         if not user in up:
    ----> 8             raise Exception('User %s not registered' % user)
          9         if pw != up[user]:
         10             raise Exception("Bad password")


        Exception: User 1 not registered


In [4]: # good call

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

Out[4]: 3

In [5]: # bad user

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


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

        Exception                                 Traceback (most recent call last)

        <ipython-input-5-b35ba770f676> in <module>()
          1 # bad user
          2
    ----> 3 foo('frank', 'bad', 1 ,2)


        <ipython-input-1-7281da65c0b8> in __call__(self, user, pw, *pos, **kw)
          6
          7         if not user in up:
    ----> 8             raise Exception('User %s not registered' % user)
          9         if pw != up[user]:
```

```
         10             raise Exception("Bad password")


        Exception: User frank not registered
```

```
In [1]: # good user, bad passwd
        foo('jack', 'nope')


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

        NameError                                 Traceback (most recent call last)

        <ipython-input-1-c358a6298bd4> in <module>()
          1 # good user, bad passwd
    ----> 2 foo('jack', 'nope')


        NameError: name 'foo' is not defined
```

```
In [6]: # works with keywords

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

Out[6]: 39
```