Homework4

February 17, 2017

1 Introduction to Python

2 Homework #4

3 Due Thursday Noon 2/23 in Courseworks

- You MUST submit on Courseworks before it closes
- Email submissions are NOT accepted

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 tips

- 'self' must be the first arg to every method
- use the 'self.' prefix to refer to instance variables or other methods inside a method

6 Problem 1 - Constraints

- suppose we want to convert between C(Celsius) and F(Fahrenheit), using the equation $9C = 5(F-32)$
- could write functions 'c2f' and 'f2c'

```
In [1]: def c2f(c):
            return( (9.*c + 5. * 32.)/5. )

        def f2c(f):
            return( 5.*(f  - 32)/9. )
```

```
In [2]: [c2f(0), c2f(100), f2c(32), f2c(212)]

Out[2]: [32.0, 212.0, 0.0, 100.0]
```

- to write f2c, we solved the equation for C, and made a function out of the other side of the equation
- to write c2f, we 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

7 define the 'setvar' method on the 'Constraint' class

- 1st arg is a variable name
  - raise a 'ValueError' if given a bad variable name
- 2nd arg is the new variable value
- if only one undefined variable is left, fire the 'constraint satisfaction'
- otherwise just record the new variable value
- do all internal computation in floating point

```
In [3]: class Constraint:
            def __init__(self, varnames, coes, total):
                self.varnames = varnames
                self.coes= [float(c) for c in coes]
                self.total = float(total)
                self.varvals = [None] * len(coes)

            def __str__(self):
                return self.__repr__()

            def __repr__(self):
                # display the status of the constraint
                # show which vars have values
                x = ' + '.join(['{}*{}(={})'.format(coe, var, val)
                                for coe,var,val in zip(self.coes, self.varnames, se
                return 'Constraint({}={})'.format(self.total, x)
```

8 tip - you may find 'dotnone' to be helpful

```
In [4]: # regular dot product, except that if one or both values in a pair is 'None
        # that term is defined to contribute 0 to the sum

        def dotnone(l1, l2):
            '''yet another dot product variant'''
            sum = 0
            for e1,e2 in zip(l1,l2):
                if not (e1 is None or e2 is None):
                    sum += e1 * e2
            return(sum)
```

```
In [5]: [dotnone([1,2,3], [4,5,6]), dotnone([1,None,3], [4,5,6]), dotnone([None,1],
```

```
Out[5]: [32, 22, 0]
```

9 Example - setup C & F constraint

- 1st init method arg is a list of the variable names,
- 2nd arg is list of coefficients for the variables
- 3rd arg is the constant total
- so, given equation 9C - 5F = -32*5, we can setup a constraint like this:
- string computed by repr method shows C & F initially have no values(=None)

```
In [6]: c = Constraint(['C', 'F'], [9,-5], -5*32)
        c
```

```
Out[6]: Constraint(-160.0=9.0*C(=None) + -5.0*F(=None))
```

10 setvar method

- 1st arg - variable name
  - raise ValueError on a bad variable name
- 2nd arg - variable value
- setvar will fire when there is one unset variable remaining. setvar will:
  - print the variable values
  - return them in a list
  - clear all variable values

```
In [7]: c.setvar('C', 100)
```

```
C = 100.0
F = 212.0


Out[7]: [100.0, 212.0]
```

```
In [8]: # bad variable name - raise an error

        c.setvar('foo', 0)
        c


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

        ValueError                                Traceback (most recent call last)

        <ipython-input-8-341c04f3a84d> in <module>()
          1 # bad variable name - raise an error
          2
    ----> 3 c.setvar('foo', 0)
          4 c


        <ipython-input-3-dc4c6ee0189c> in setvar(self, varname, val)
         18     def setvar(self, varname, val):
         19         if not varname in self.varnames:
    ---> 20             raise ValueError('varname ' + varname + " is not defined in
         21         n = self.varnames.index(varname)
         22         self.varvals[n] = float(val)


        ValueError: varname foo is not defined in ['C', 'F']


In [9]: c.setvar('F', 212)

C = 100.0
F = 212.0


Out[9]: [100.0, 212.0]
```

11 more complex example

- 5 constraint variables

```
In [10]: c2 = Constraint(['x0', 'x1', 'x2', 'x3', 'x4'], range(5), 1)
         c2

Out[10]: Constraint(1.0=0.0*x0(=None) + 1.0*x1(=None) + 2.0*x2(=None) + 3.0*x3(=Nor

In [11]: c2.setvar('x1', 10)
         c2

Out[11]: Constraint(1.0=0.0*x0(=None) + 1.0*x1(=10.0) + 2.0*x2(=None) + 3.0*x3(=Nor
```

```
In [12]: c2.setvar('x0', 0)
         c2

Out[12]: Constraint(1.0=0.0*x0(=0.0) + 1.0*x1(=10.0) + 2.0*x2(=None) + 3.0*x3(=None

In [13]: # x2

         c2.setvar('x2',20)
         c2

Out[13]: Constraint(1.0=0.0*x0(=0.0) + 1.0*x1(=10.0) + 2.0*x2(=20.0) + 3.0*x3(=None

In [14]: # only two unset vars left, so setting x3 or x4
         # will fire the constraints

         c2.setvar('x4', 30)

x0 = 0.0
x1 = 10.0
x2 = 20.0
x3 = -56.333333333333336
x4 = 30.0


Out[14]: [0.0, 10.0, 20.0, -56.333333333333336, 30.0]
```

12 sketchpad(1962)

- IMHO, sketchpad is the greatest CS Phd thesis ever.
- among other things, Ivan Sutherland invented constraint systems, interactive computer graphics, CAD, object oriented programming, and visual programming
- the computer he used had 33K of memory!
- if you have a few minutes sometime, watch sketchpad video from summer 1962

13 Problem 2 - Hamlet

- Python is very popular in 'Digital Humanities'
- MIT has the complete works of Shakespeare in a simple html format
- You will do a simple analysis of Hamlet by reading the html file, and doing pattern matching
- The goal is to return a list of the line count, total number of 'speeches'(look at the file format), and a dict showing the number of 'speeches' each character gives
- Your program should read directly from the url given below, but you may want to download a copy to examine the structure of the file.
- remember that urllib.request returns 'byte arrays', not strings
- there are at least three ways to do this - your choice
  - use string methods like 'find'
  - use regular expressions

  - use Beautiful Soup(won't get a line count with this method)

- here's a short sample of the file

```
<A NAME=speech25><b>HORATIO</b></a>
<blockquote>
<A NAME=1.1.37>Tush, tush, 'twill not appear.</A><br>
</blockquote>

<A NAME=speech26><b>BERNARDO</b></a>
<blockquote>
<A NAME=1.1.38>Sit down awhile;</A><br>
<A NAME=1.1.39>And let us once again assail your ears,</A><br>
<A NAME=1.1.40>That are so fortified against our story</A><br>
<A NAME=1.1.41>What we have two nights seen.</A><br>
</blockquote>

<A NAME=speech27><b>HORATIO</b></a>
<blockquote>
<A NAME=1.1.42>Well, sit we down,</A><br>
<A NAME=1.1.43>And let us hear Bernardo speak of this.</A><br>
</blockquote>

<A NAME=speech28><b>BERNARDO</b></a>
<blockquote>
<A NAME=1.1.44>Last night of all,</A><br>
<A NAME=1.1.45>When yond same star that's westward from the pole</A><br>
<A NAME=1.1.46>Had made his course to illume that part of heaven</A><br>
<A NAME=1.1.47>Where now it burns, Marcellus and myself,</A><br>
<A NAME=1.1.48>The bell then beating one,--</A><br>
<p><i>Enter Ghost</i></p>
</blockquote>

<A NAME=speech29><b>MARCELLUS</b></a>
<blockquote>
<A NAME=1.1.49>Peace, break thee off; look, where it comes again!</A><br>
</blockquote>

<A NAME=speech30><b>BERNARDO</b></a>
<blockquote>
<A NAME=1.1.50>In the same figure, like the king that's dead.</A><br>
</blockquote>
```

```
In [15]: # use this url for hamlet - do not hit MIT directly
         # break up long line

         import urllib.request
         import collections
```

```
import re
import bs4
import lxml

url = 'https://courseworks.columbia.edu/access/content/group/'
url += 'COMSW3101_002_2015_3/data/hamlet.html'
```

In [47]:
```
hamlet(url)
```

Out[47]:
```
[8881,
 1150,
 defaultdict(int,
             {'All': 4,
              'BERNARDO': 23,
              'CORNELIUS': 1,
              'Captain': 7,
              'Danes': 3,
              'FRANCISCO': 8,
              'First Ambassador': 1,
              'First Clown': 33,
              'First Player': 8,
              'First Priest': 2,
              'First Sailor': 2,
              'GUILDENSTERN': 33,
              'Gentleman': 3,
              'Ghost': 14,
              'HAMLET': 359,
              'HORATIO': 112,
              'KING CLAUDIUS': 102,
              'LAERTES': 62,
              'LORD POLONIUS': 86,
              'LUCIANUS': 1,
              'Lord': 3,
              'MARCELLUS': 36,
              'Messenger': 2,
              'OPHELIA': 58,
              'OSRIC': 25,
              'PRINCE FORTINBRAS': 6,
              'Player King': 4,
              'Player Queen': 5,
              'Prologue': 1,
              'QUEEN GERTRUDE': 69,
              'REYNALDO': 13,
              'ROSENCRANTZ': 49,
              'Second Clown': 12,
              'Servant': 1,
              'VOLTIMAND': 2})]
```

14 Problem 3 - Interval

- implement a class 'Interval', that does 'interval arithmetic' and defines '+' and '*' operators
- an interval consists of a min and max value. use instance variable names 'imin', 'imax' to avoid confusion with 'min' and 'max' functions
- let 'i' and 'i2' be intervals
- `i + i2 represents a new interval, where the new imin and imax is the min and max of (x + x2), where i.imin <= x <= i.imax and i2.imin <= x2 <= i2.imax`
  - define `__add__` method
- `i * i2 represents a new interval, where the new imin and imax is the min and max of (x * x2), where i.imin <= x <= i.imax and i2.imin <= x2 <= i2.imax`
  - define `__mul__` method
- adding intervals is easy
- multiplying intervals - think for a second
- should be able to add or multiply by a scalar(an integer) on the right, by checking the type of the argument to `__add__ and __mul__`
  - let i be an Interval, s a scalar(integer)
    - `i + s is the same as i + Interval(s, s)`
    - `i * s is the same as i * Interval(s, s)`
- an interval should print as `Interval<imin, imax>`
- use only integers, no floats

```
In [29]: i = Interval(-1,6)
         i2 = Interval(5, 13)
         i3 = Interval(10,10)

         [i, i2, i3]

Out[29]: [Interval<-1, 6>, Interval<5, 13>, Interval<10, 10>]

In [23]: i + 10

Out[23]: Interval<9, 16>

In [24]: i + i2

Out[24]: Interval<4, 19>

In [25]: i + i3

Out[25]: Interval<9, 16>

In [26]: i * 10
```

```
Out[26]: Interval<-10, 60>

In [27]: i * i2

Out[27]: Interval<-13, 78>

In [28]: i * i3

Out[28]: Interval<-10, 60>
```

15 Problem 4 & 5 - vending machine

- use objects to simulate a vending machine
- money is in units of cents

16 class venditem represents a type of item for sale

- has three instance variables
  - name, price, quantity
- define four methods
  - `method __init__` loads data into the instance variables
    - def __init__(self, name, price, quantity):
  - `method __repr__`(self)
    - controls how venditem prints
    - use string format method
      - '{} {}'.format(arg, arg2)
    - see examples below
  - `method __str__`(self)
    - just call `__repr__` for string to return
  - method sale(self)
    - decrement the quantity

```
In [31]: # __repr__ method shows object status

         vi = venditem('coke', 95, 3)
         vi2 = venditem('pepsi', 110, 1)

         [vi, vi2]

Out[31]: [venditem(name='coke', price=95, quantity=3),
          venditem(name='pepsi', price=110, quantity=1)]

In [32]: # sale method decrements quantity instance variable

         vi.sale()
         vi
```

```
Out[32]: venditem(name='coke', price=95, quantity=2)

In [33]: # note you can access instance variables directly:

         [vi.name, vi2.name, vi.price, vi.quantity, vi2.quantity]

Out[33]: ['coke', 'pepsi', 95, 2, 1]

In [34]: # can set same way

         vi.quantity = 2
         vi.quantity

Out[34]: 2
```

17 class vendmachine

- vendmachine has two instance variables
  - 'cash' - the amount of money the machine has collected from item sales
  - 'items' - a dictionary, where keys are the name of an item, and the values are the venditem object
- define three methods(log method is done for you)
  - `__init__`(self, stock)
    - 1st arg - stock is a list of venditems, which represents what is loaded in the machine
    - items dictionary should be constructed from stock
    - cash should be initialized to 0
  - buy(self, name, money)
    - 'name' is 'coke', 'pepsi', etc
    - money is how much money the customer deposited for the purchase
    - four cases
      - customer asks for an item not carried
      - customer asks for an item whose quantity is 0 - out of stock
      - customer doesn't put in enough money for the item
      - everything ok, sell the item, decrement item quantity
    - 'buy' return value should refund any money owed the customer
      - money not applied to an item sale
      - excess money deposited for an item sale
    - log each buy case, using 'log' method below
    - see examples below
  - status(self)
    - prints the amount of cash collected, and each of the items in stock

```
In [ ]: import time

        class vendmachine:
```

```
            def log(self, msg, name):
                t = time.strftime('%X %x %Z - ')
                msg = t + msg + ': ' + name
                print(msg)
```

```
In [38]: # make stock for sale and load vendmachine

         vi = venditem('coke', 95, 3)
         vi2 = venditem('pepsi', 110, 1)
         vi3 = venditem('peanut M&Ms', 100, 2)
         stock = [vi, vi2, vi3]

         vm = vendmachine(stock)
         vm.status()
```

```
cash collected: 0
venditem(name='peanut M&Ms', price=100, quantity=2)
venditem(name='pepsi', price=110, quantity=1)
venditem(name='coke', price=95, quantity=3)
```

```
In [39]: vm.buy('coke', 45)
```

```
10:34:53 02/17/17 EST - insufficient funds for: coke
```

```
Out[39]: 45
```

```
In [40]: vm.buy('pepsi', 200)
```

```
10:34:55 02/17/17 EST - sold : pepsi
```

```
Out[40]: 90
```

```
In [41]: vm.status()
```

```
cash collected: 110
venditem(name='peanut M&Ms', price=100, quantity=2)
venditem(name='pepsi', price=110, quantity=0)
venditem(name='coke', price=95, quantity=3)
```

```
In [42]: vm.buy('pepsi', 200)
```

```
10:35:07 02/17/17 EST - out of stock: pepsi
```

```
Out[42]: 200

In [44]: vm.buy('mountain dew', 200)

10:35:33 02/17/17 EST - dont carry it: mountain dew


Out[44]: 200

In [45]: vm.buy('coke', 100)

10:35:37 02/17/17 EST - sold : coke


Out[45]: 5

In [46]: vm.status()

cash collected: 205
venditem(name='peanut M&Ms', price=100, quantity=2)
venditem(name='pepsi', price=110, quantity=0)
venditem(name='coke', price=95, quantity=2)
```