homework-6

May 2, 2016

1 Introduction to Python

2 Homework #6

3 Due Monday May 9, 6pm

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 have already done four homeworks and are happy with them, you do NOT need to do this one

6 Problem 1 - stars

7 get stars data

- download and unpack stars.zip somewhere in your file system
- set 'basedir' to location of 'YOURPATH/stars/near'

```
In [251]: basedir = '/tmp/stars/near'
```

8 write loadFileProperties

- each line in file is in 'key=value' format
- value is either a float or a quoted string
- convert 'float strings' into floats

```
In [249]: sunpath = os.path.join(basedir, 'Sun')
```

```
          with open(sunpath) as f:
              for line in f:
                  print(line.strip())
```

```
ApparentMagnitude=-26.72
EffectiveTemperature=5.78e3
Constellation="Virgo"
Gravity=898.79
SpectralClass="G2V"
EscapeVelocity=1.3814e6
AlphanumericName="Sun"
```

```
In [250]: # convert value float strings to floats
          # string values should only have one set of quotes -
          # should have
          # 'SpectralClass': 'G2V'
          # not
          # 'SpectralClass': '"G2V"'

          loadFileProperties(sunpath)
```

```
Out[250]: {'AlphanumericName': 'Sun',
           'ApparentMagnitude': -26.72,
           'Constellation': 'Virgo',
           'EffectiveTemperature': 5780.0,
           'EscapeVelocity': 1381400.0,
           'Gravity': 898.79,
           'SpectralClass': 'G2V'}
```

9 write stars class

- `__init__()` method
  - makes a sqlite3 database(in memory)
  - defines a star table in SQL
  - loads all the stars in stars/near, using loadFileProperties, into the DB
- query() method just hits SQL DB
- sqlite3 doc

```
In [252]: import os
          import sqlite3

          class stars:

              def __init__(self, basedir):
                  # makes a database that lives in memory
                  self.con = sqlite3.connect(':memory:')

                  fields = ['AlphanumericName', 'ApparentMagnitude', 'Constellation',
                      'EffectiveTemperature', 'EscapeVelocity',
                      'Gravity', 'SpectralClass']
                  sqlTypes = ['text', 'real', 'text', 'real', 'real', 'real', 'text']

                  # make star table in DB

                  # your code

                  # load star data into DB star table
```

```
                # your code


            def query(self, q):
                return self.con.execute(q)
```

```
In [253]: # print how many stars you load - should get 55

          s=stars(basedir)
```

```
loaded 55 stars
```

```
In [254]: list(s.query('select AlphanumericName,Gravity,EscapeVelocity from star where Gravity>2400'))
```

```
Out[254]: [('GJ 1061', 2800.0, 1200000.0),
           ('GJ 1245 A', 2800.0, 1200000.0),
           ('Gl 15 B', 3000.0, 1200000.0),
           ('Gl 473 A', 2800.0, 1200000.0),
           ('Gl 752 B', 2600.0, 1200000.0),
           ('Gl 860 B', 3000.0, 1200000.0),
           ('Gl 866 A', 2600.0, 1200000.0),
           ('Gl 905', 3000.0, 1200000.0),
           ('HIP 5643', 2800.0, 1200000.0),
           ('Luyten 726-8 A', 2800.0, 1200000.0),
           ('Luyten 726-8 B', 2800.0, 1200000.0),
           ("Luyten's Star", 2600.0, 1200000.0),
           ('Proxima Centauri', 2600.0, 1200000.0),
           ('Sirius B', 12000000.0, 15000000.0),
           ('Wolf 359', 3000.0, 1200000.0)]
```

```
In [255]: list(s.query('select avg(Gravity), avg(EscapeVelocity) from star where Gravity>2400'))
```

```
Out[255]: [(802613.3333333334, 2120000.0)]
```

10 Problem 2 - interleave

- takes two strings and computes every possible interleave
- an interleave uses all the characters from each string once, and the order of each string is preserved in the interleaved string
- each input string is in alphabetical order, with no duplicated characters
- first arg is all lower case, second arg is all upper case
- many ways to do this
- one straightforward approach is to concat the two strings, and apply itertools.permutations(). this will generate a list containing all the interleaved strings, plus bogus ones. just filter out the bogus strings, and return what's left
- in a bogus string the lower case and/or upper case letters are not in the original order

```
In [242]: interleave('abcde', 'X')
```

```
Out[242]: ['abcdeX', 'abcdXe', 'abcXde', 'abXcde', 'aXbcde', 'Xabcde']
```

```
In [241]: # 'baXY', 'YXab', 'baYX' are bogus, because the order of
          # one or both of the lower and upper case strings has not
          # been maintained

          interleave('ab', 'XY')
```

```
Out[241]: ['abXY', 'aXbY', 'aXYb', 'XabY', 'XaYb', 'XYab']

In [240]: interleave('abc', 'XYZ')

Out[240]: ['abcXYZ',
           'abXcYZ',
           'abXYcZ',
           'abXYZc',
           'aXbcYZ',
           'aXbYcZ',
           'aXbYZc',
           'aXYbcZ',
           'aXYbZc',
           'aXYZbc',
           'XabcYZ',
           'XabYcZ',
           'XabYZc',
           'XaYbcZ',
           'XaYbZc',
           'XaYZbc',
           'XYabcZ',
           'XYabZc',
           'XYaZbc',
           'XYZabc']
```

11 Problem 3 - zipfiles

- Extract data directly from a zipfile without unpacking it
- use zipfile module doc. only need three things
  - zipfile.ZipFile()
  - zipfile.namelist()
  - zipfile.open()
- extract info from python's help documentation
  - on mac, clicking on the zip link will automatically unpack, which we don't want. instead right click on the link, and do 'Download Linked File as...' instead
- zip file to download zip of python docs in plain text format

```
In [283]: # set to where ever you place the python docs zip

          zfile = '/Users/lstead/Downloads/python-3.5.1-docs-text.zip'
```

12 write help class

- `__init__()` method should read in all the help files and save them in the help object. remember to convert to unicode, and remove new lines
  - hits() method - checks all file lines for match, and return hit count
  - grep() method - print file name, line number, and line of each match
  - files() method - return dict of file/# of hits

```
In [258]: h = help(zfile)

reading 477 files
```

```
In [131]: h.files('zipfiles')

Out[131]: defaultdict(int,
                      {'python-3.5.1-docs-text/distutils/apiref.txt': 1,
                       'python-3.5.1-docs-text/library/warnings.txt': 1,
                       'python-3.5.1-docs-text/reference/import.txt': 1,
                       'python-3.5.1-docs-text/using/cmdline.txt': 3,
                       'python-3.5.1-docs-text/whatsnew/3.2.txt': 1,
                       'python-3.5.1-docs-text/whatsnew/changelog.txt': 1})

In [118]: [h.hits(w) for w in ['rpartition', 'linux','universal', 'mode', 'zipfile', 'zipfiles']]

Out[118]: [10, 41, 67, 1103, 78, 8]

In [99]: h.grep('rpartition')

python-3.5.1-docs-text/whatsnew/2.7.txt:917: * The "split()", "replace()", "rindex()", "rpartition()", a
python-3.5.1-docs-text/library/stdtypes.txt:1519: str.rpartition(sep)
python-3.5.1-docs-text/library/stdtypes.txt:2275: bytes.rpartition(sep)
python-3.5.1-docs-text/library/stdtypes.txt:2276: bytearray.rpartition(sep)
python-3.5.1-docs-text/whatsnew/2.5.txt:976: "rpartition(sep)" methods that simplify a common use case.
python-3.5.1-docs-text/whatsnew/2.5.txt:985: "rpartition(sep)" also returns a 3-tuple but starts searchi
python-3.5.1-docs-text/whatsnew/2.5.txt:996: >>> 'www.python.org'.rpartition('.')
python-3.5.1-docs-text/whatsnew/2.5.txt:998: >>> 'www.python.org'.rpartition(':')
python-3.5.1-docs-text/whatsnew/3.2.txt:2368: extension = name.rpartition('.')[2]
python-3.5.1-docs-text/whatsnew/3.2.txt:2409: also used by "rfind()", "rindex()", "rsplit()" and "rparti

In [103]: h.grep('zipfiles')

python-3.5.1-docs-text/library/warnings.txt:322: used to support displaying source for modules found in
python-3.5.1-docs-text/distutils/apiref.txt:794: as tarballs or zipfiles.
python-3.5.1-docs-text/using/cmdline.txt:83: and zipfiles that are passed to the interpreter as the scri
python-3.5.1-docs-text/using/cmdline.txt:430: may refer to zipfiles containing pure Python modules (in e
python-3.5.1-docs-text/using/cmdline.txt:432: zipfiles.
python-3.5.1-docs-text/whatsnew/changelog.txt:2278: * Issue #20078: Reading malformed zipfiles no longer
python-3.5.1-docs-text/whatsnew/3.2.txt:1531: zipfiles, uncompressed tarfiles, gzipped tarfiles, and bzi
python-3.5.1-docs-text/reference/import.txt:653: libraries) from zipfiles.

In [168]: h.files('zipfiles')

Out[168]: defaultdict(int,
                      {'python-3.5.1-docs-text/distutils/apiref.txt': 1,
                       'python-3.5.1-docs-text/library/warnings.txt': 1,
                       'python-3.5.1-docs-text/reference/import.txt': 1,
                       'python-3.5.1-docs-text/using/cmdline.txt': 3,
                       'python-3.5.1-docs-text/whatsnew/3.2.txt': 1,
                       'python-3.5.1-docs-text/whatsnew/changelog.txt': 1})
```

13 Problem 4 - sumTrianglePath

14 Write str2lists

- converts string form of triangle data into nested lists
- homework-6.py has the data in ascii

```
In [274]: s1='''
             3
            7 4
           2 4 6
          '''
          x1= str2lists(s1)
          x1
```

```
Out[274]: [[3], [7, 4], [2, 4, 6]]
```

```
In [281]: s2='''
             3
            7 4
           2 4 6
          8 5 9 3
          '''
          x2 = str2lists(s2)
          x2
```

```
Out[281]: [[3], [7, 4], [2, 4, 6], [8, 5, 9, 3]]
```

```
In [275]: s3 = '''
                            75
                           95 64
                          17 47 82
                         18 35 87 10
                        20 04 82 47 65
                       19 01 23 75 03 34
                      88 02 77 73 07 63 67
                     99 65 04 28 06 16 70 92
                    41 41 26 56 83 40 80 70 33
                   41 48 72 33 47 32 37 16 94 29
                  53 71 44 65 25 43 91 52 97 51 14
                 70 11 33 28 77 73 17 78 39 68 17 57
                91 71 52 38 17 14 91 43 58 50 27 29 48
              63 66 04 68 89 53 67 30 73 16 69 87 40 31
          04 62 98 27 23 09 70 98 73 93 38 53 60 04 23
          '''

          x3 = str2lists(s3)
          x3
```

```
Out[275]: [[75],
           [95, 64],
           [17, 47, 82],
           [18, 35, 87, 10],
           [20, 4, 82, 47, 65],
           [19, 1, 23, 75, 3, 34],
           [88, 2, 77, 73, 7, 63, 67],
           [99, 65, 4, 28, 6, 16, 70, 92],
           [41, 41, 26, 56, 83, 40, 80, 70, 33],
           [41, 48, 72, 33, 47, 32, 37, 16, 94, 29],
           [53, 71, 44, 65, 25, 43, 91, 52, 97, 51, 14],
           [70, 11, 33, 28, 77, 73, 17, 78, 39, 68, 17, 57],
           [91, 71, 52, 38, 17, 14, 91, 43, 58, 50, 27, 29, 48],
           [63, 66, 4, 68, 89, 53, 67, 30, 73, 16, 69, 87, 40, 31],
           [4, 62, 98, 27, 23, 9, 70, 98, 73, 93, 38, 53, 60, 4, 23]]
```

15 Write sumTrianglePath

- takes nested list triangle represention and returns the max path
- starting at the top, and moving down to adjacent numbers on the row below, find the path with the maximum sum
- hint: this is very simple
  - keep merging the last row of the triangle into the 2nd to last row, until you are done
  - you may find it convenient to make a copy of the input, use copy.deepcopy
  - don't need recursion
- in triangle below, the max sum from top to bottom is 3+7+4+9=23

```
   3
  7 4
 2 4 6
8 5 9 3
```

```
In [282]: [sumTrianglePath(x) for x in [x1, x2, x3]]
```

```
Out[282]: [14, 23, 1074]
```

16 Problem 5 - Conway's game of life

- the `__init__()` method takes as input an n x n numpy array of 1s and 0s, representing different cell states. A cell is considered alive if it is of state 1, and dead otherwise. The neighbors of a cell are any of the cells that are adjacent to it, 3 at a corner, 5 on an edge, 8 in the interior.
- the `__next__()` method operates on the current board, and returns the next state of the board as determined by the following rules:
  - If a cell is alive, and it has exactly two or three neighbors that is also alive, then in the next turn, it will remain alive.
  - If a cell is alive and it has less than two or more than three alive neighbors, it will die.
  - If a cell is dead and it has exactly three neighbors which are alive, then next turn it will come back to life. Otherwise, it remains dead.
- use 2d numpy array internally
- you must complete the `__next__()` and countAliveNeighbors() methods
- many interesting patterns have been discovered
- wiki article

```
In [263]: import numpy as np
```

```
          class life:
              def __init__(self, board):
                  # current board - will be updated by __next__
                  self.current = np.array(board)
                  self.n = len(board)

              def run(self, n):
                  self.plife()
                  for j in range(n):
                      next(self)
                      self.plife()

              def plife(self):
                  s = []
```

```
                for row in range(self.n):
                    for col in range(self.n):
                        s.append('1' if self.current[row,col] else '0')
                    s.append('\n')
                print(''.join(s))

            def __next__(self):
                outputBoard = self.current.copy()

                # compute and store new board in outputBoard

                # Your code here

                self.current = outputBoard
                return outputBoard

            def countAliveNeighbors(self, i, j):
                # don't go off the board on edges
                count = 0

                # Your code here

                return count
```

In [261]:
```
# homework-6.py has the data in ascii

# stable, doesn't change

block = life([[0,0,0,0],
              [0,1,1,0],
              [0,1,1,0],
              [0,0,0,0]])
```

In [262]:
```
block.run(2)
```

```
0000
0110
0110
0000

0000
0110
0110
0000

0000
0110
0110
0000
```

In [5]:
```
# aternates between two patterns

blinker = life([[0, 1, 0 ],
                [0, 1, 0],
                [0, 1, 0]])
```

```
        blinker.run(4)
```

```
010
010
010

000
111
000

010
010
010

000
111
000

010
010
010
```

```
In [16]: toad = life(np.array([[0, 0, 0, 0, 0, 0],
                 [0, 0, 0, 0, 0, 0],
                 [0, 0, 1, 1, 1, 0],
                 [0, 1, 1, 1, 0, 0],
                 [0, 0, 0, 0, 0, 0],
                 [0, 0, 0, 0, 0, 0]]))

         toad.run(2)
```

```
000000
000000
001110
011100
000000
000000

000000
000100
010010
010010
001000
000000

000000
000000
001110
011100
000000
000000
```

```
In [264]: everybodydies = life(np.array([
                      [1,1,1,0,1],
```

```
                    [1,0,0,1,0],
                    [0,1,0,0,1],
                    [1,1,0,1,0],
                    [1,0,1,0,1],
                    ]))
everybodydies.run(16)
```

```
11101
10010
01001
11010
10101

11110
10011
01011
10011
10110

11111
10000
11000
10000
01111

11110
00010
11000
10010
01110

01110
00010
11100
10010
01110

00110
10010
11110
10010
01110

00110
10001
10011
10001
01110

00010
01101
11011
10001
01110
```

```
00110
11001
10001
10001
01110

01110
11101
10011
10101
01110

10010
10001
10001
10001
01110

00000
11011
11011
10101
01110

00000
11011
00000
10001
01110

00000
00000
11011
01110
01110

00000
00000
11011
00000
01010

00000
00000
00000
11011
00000

00000
00000
00000
00000
00000
```

```
In [6]: fig8 = life(np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]))

        fig8.run(8)
```

```
000000000000
000000000000
000000000000
000111000000
000111000000
000111000000
000000111000
000000111000
000000111000
000000000000
000000000000
000000000000

000000000000
000000000000
000010000000
000101000000
001000100000
000100010000
000010001000
000001000100
000000101000
000000010000
000000000000
000000000000

000000000000
000000000000
000010000000
000111000000
001110100000
000100010000
000010001000
000001011100
000000111000
000000010000
000000000000
000000000000
```

000000000000
000000000000
000111000000
001000000000
001000100000
001001010000
000010100100
000001000100
000000000100
000000111000
000000000000
000000000000

000000000000
000010000000
000110000000
001011000000
011100100000
000101010000
000010101000
000001001110
000000110100
000000011000
000000010000
000000000000

000000000000
000110000000
000000000000
010001000000
010000100000
000101010000
000010101000
000001000010
000000100010
000000000000
000000011000
000000000000

000000000000
000000000000
000010000000
000000000000
001011100000
000011010000
000010110000
000001110100
000000000000
000000010000
000000000000
000000000000

000000000000
000000000000

000000000000
000110000000
000110100000
000000010000
000010000000
000001011000
000000011000
000000000000
000000000000
000000000000

000000000000
000000000000
000000000000
000111000000
000111000000
000111000000
000000111000
000000111000
000000111000
000000000000
000000000000
000000000000