homework-2

September 20, 2015

1 Introduction to Python

2 Homework #2

3 Due Friday, Sept 25, 1:45pm in Courseworks

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.

4.1 Tips

4.1.1 Generators

```
In [139]: # you can terminate a generator by using 'return',
          # or falling off the end of the generator
          def g1():
              yield(1)
              yield(2)
              return
              yield(3)

          def g2():
              yield(1)
              yield(2)

          # an easy way to get the elements from a
          # FINITE length generator will return is to use 'list'
          print(list(g1()))
          print(list(g2()))
```

```
[1, 2]
[1, 2]
```

```
In [140]: # if a generator calls a 2nd generator for elements,
          # and the 2nd generator finishes, the 1st one will finish as well
          def g3(g):
              while True:
                  yield(next(g))

          for i in g3(g2()):
              print(i)
```

```
1
2
```

4.1.2 String

```
In [40]: # split will "tokenize", and get rid of white space

         ' foo     bar zip         zap  '.split()

Out[40]: ['foo', 'bar', 'zip', 'zap']

In [51]: # replace can remove unwanted chars
         'dont, want, the, commas,'.replace(',','')

Out[51]: 'dont want the commas'

In [65]: # convert strings to numbers

         [int('2343'), float('2.34')]

Out[65]: [2343, 2.34]
```

4.1.3 Arithmetic

```
In [1]: # integer quotient, integer remainder, floating divide
        [14//5, 14 % 5, 14 / 5]

Out[1]: [2, 4, 2.8]

In [149]: # 'bin' function converts an int into a binary string
          [bin(11), bin(234)]

Out[149]: ['0b1011', '0b11101010']

In [24]: # 1 and 0 function as True and False in an 'if' statement
         [True if 1 else False, True if 0 else False]

Out[24]: [True, False]
```

5 Problem 1a - decimals

- define a 'decimals' generator function, that 'generates' the decimal digits of 1/n, where n is an integer greater than 1
- if the decimal expansion terminates, like $1/8 = .125$, the generator should terminate. otherwise, like for 1/3=.333..., the generator should never stop
- use long division to compute the expansion - it is very simple

```
In [72]: list(decimals(8))

Out[72]: [1, 2, 5]

In [73]: d = decimals(3)
         [next(d) for x in range(10)]

Out[73]: [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

6 Problem 1b - genlimit

- define ‘genlimit(g, limit)’, which generates at most ‘limit’ number of values from a generator ‘g’

```
In [75]: print(list(genlimit(decimals(3), 5)))
         print(list(genlimit(decimals(8), 5)))
```

```
[3, 3, 3, 3, 3]
[1, 2, 5]
```

7 Problem 2 - Deal With Repeated Decimals

- genlimit is useful, but never sure what we’re missing with an arbitrary limit
- since 1/n is a rational number, its decimal expansion must eventually repeat(unlike irrational numbers like PI)
- write ‘decimals2’, a variant of ‘decimals’
- if the decimal expansion is finite, it should just return the finite set of digits
- if the decimal expansion repeats, it should return the digits up to the point it starts repeating. then the final yield should be the repeating sequence of digits

```
In [142]: import textwrap

          for j in range(2, 30):
              # ugly hack needed because lines don’t wrap in pdf version
              d = list(decimals2(j))
              print('   Expansion of 1/' + str(j) + ':')
              print( textwrap.fill(str(d), 80))
```

```
Expansion of 1/2:
[5]
   Expansion of 1/3:
[3, [3]]
   Expansion of 1/4:
[2, 5]
   Expansion of 1/5:
[2]
   Expansion of 1/6:
[1, 6, [6]]
   Expansion of 1/7:
[1, 4, 2, 8, 5, 7, [1, 4, 2, 8, 5, 7]]
   Expansion of 1/8:
[1, 2, 5]
   Expansion of 1/9:
[1, [1]]
   Expansion of 1/10:
[1]
   Expansion of 1/11:
[0, 9, [0, 9]]
   Expansion of 1/12:
[0, 8, 3, [3]]
   Expansion of 1/13:
[0, 7, 6, 9, 2, 3, [0, 7, 6, 9, 2, 3]]
   Expansion of 1/14:
[0, 7, 1, 4, 2, 8, 5, [7, 1, 4, 2, 8, 5]]
   Expansion of 1/15:
```

```
[0, 6, [6]]
   Expansion of 1/16:
[0, 6, 2, 5]
   Expansion of 1/17:
[0, 5, 8, 8, 2, 3, 5, 2, 9, 4, 1, 1, 7, 6, 4, 7, [0, 5, 8, 8, 2, 3, 5, 2, 9, 4,
1, 1, 7, 6, 4, 7]]
   Expansion of 1/18:
[0, 5, [5]]
   Expansion of 1/19:
[0, 5, 2, 6, 3, 1, 5, 7, 8, 9, 4, 7, 3, 6, 8, 4, 2, 1, [0, 5, 2, 6, 3, 1, 5, 7,
8, 9, 4, 7, 3, 6, 8, 4, 2, 1]]
   Expansion of 1/20:
[0, 5]
   Expansion of 1/21:
[0, 4, 7, 6, 1, 9, [0, 4, 7, 6, 1, 9]]
   Expansion of 1/22:
[0, 4, 5, [4, 5]]
   Expansion of 1/23:
[0, 4, 3, 4, 7, 8, 2, 6, 0, 8, 6, 9, 5, 6, 5, 2, 1, 7, 3, 9, 1, 3, [0, 4, 3, 4,
7, 8, 2, 6, 0, 8, 6, 9, 5, 6, 5, 2, 1, 7, 3, 9, 1, 3]]
   Expansion of 1/24:
[0, 4, 1, 6, [6]]
   Expansion of 1/25:
[0, 4]
   Expansion of 1/26:
[0, 3, 8, 4, 6, 1, 5, [3, 8, 4, 6, 1, 5]]
   Expansion of 1/27:
[0, 3, 7, [0, 3, 7]]
   Expansion of 1/28:
[0, 3, 5, 7, 1, 4, 2, 8, [5, 7, 1, 4, 2, 8]]
   Expansion of 1/29:
[0, 3, 4, 4, 8, 2, 7, 5, 8, 6, 2, 0, 6, 8, 9, 6, 5, 5, 1, 7, 2, 4, 1, 3, 7, 9,
3, 1, [0, 3, 4, 4, 8, 2, 7, 5, 8, 6, 2, 0, 6, 8, 9, 6, 5, 5, 1, 7, 2, 4, 1, 3,
7, 9, 3, 1]]
```

8 Problem 3a - select

- define a function ‘select(l, selectors)’, where ‘l’ and ‘selectors’ lists of the same length
- ‘select’ returns new list which consists of the elements of l that have a True value in the corresponding selectors element

```
In [79]: select(range(5), [0, 1, '', "foo", True])
```

```
Out[79]: [1, 3, 4]
```

9 Problem 3b - nbitIntToDigits

- define a function ‘nbitIntToDigits(i, n)’
- returns a list of the digits(ints, not strings) in a binary representation of ‘i’
- pad with ’0’s on the left if needed to generate n bits

```
In [81]: [nbitIntToDigits(3, 2), nbitIntToDigits(3, 6), nbitIntToDigits(11, 4)]
```

```
Out[81]: [[1, 1], [0, 0, 0, 0, 1, 1], [1, 0, 1, 1]]
```

10 Problem 3c - powerSet

- define a function ‘powerSets(l)’ that returns all possible subsets of the elements of l, including the empty list and the original list
- for a list of length n, how many elements are in the powerSet list?

```
In [83]: powerSet(['avery', 'math', 'butler'])

Out[83]: [[],
          ['butler'],
          ['math'],
          ['math', 'butler'],
          ['avery'],
          ['avery', 'butler'],
          ['avery', 'math'],
          ['avery', 'math', 'butler']]
```

```
In [84]: powerSet(['avery', 'math', 'butler', 'dodge'])

Out[84]: [[],
          ['dodge'],
          ['butler'],
          ['butler', 'dodge'],
          ['math'],
          ['math', 'dodge'],
          ['math', 'butler'],
          ['math', 'butler', 'dodge'],
          ['avery'],
          ['avery', 'dodge'],
          ['avery', 'butler'],
          ['avery', 'butler', 'dodge'],
          ['avery', 'math'],
          ['avery', 'math', 'dodge'],
          ['avery', 'math', 'butler'],
          ['avery', 'math', 'butler', 'dodge']]
```

11 Problem 4 - Decrypting Government Data

- Your job is to summarize this gov data about oil consumation
- The format of the file rather bizzare - note that each line has data for two months, in two different years! (Plus I had to edit the file to make it parseable)
- Fortunately, Python is great for untangling and manipulating data.
- Write a generator that produces a summary line for a year’s data on each ‘next’ call
- The generator should read the lines of the oil file in a lazy fashion - it should only read 13 lines for every two years of output. Note a loop can have any number of ‘yield’ calls in it.
- Your generator probably wants to throw away the first 7 lines when it starts.
- The summary line produced should include the total consumption over the year(sum up the per month consumption), and the min, max, and avg price for the year.
- In addition to the ‘oil’ generator function, my solution had a separate helper function, ‘report’ to make the summary string:

- def report(years, data, leftside):
  - years is the two years the 13 lines cover, like [‘2014’, ‘2013’]
  - data is the 13 lines covering two years
  - leftside is True if i’m extracting the ‘left’ half of the data, false if extracting from the right.
- Download the ‘oil.txt’ file from Courseworks/week2

```
In [143]: # change this path to where you put 'oil.txt' on your machine
          path = '/tmp/oil.txt'
          o = oil(path)

In [144]: next(o)

Out[144]: '2014: quan: total=2700903 prices: max = 97.810000 min = 73.640000 avg = 91.282500'

In [145]: next(o)

Out[145]: '2013: quan: total=2813771 prices: max = 102.000000 min = 91.330000 avg = 96.937500'

In [146]: for s in list(o):
              print(s)

2012: quan: total=3097408 prices: max = 109.620000 min = 93.670000 avg = 101.019167
2011: quan: total=3321917 prices: max = 108.740000 min = 84.500000 avg = 99.730833
2010: quan: total=3377077 prices: max = 79.790000 min = 72.090000 avg = 74.700000
2009: quan: total=3314787 prices: max = 73.140000 min = 39.140000 avg = 57.165833
2008: quan: total=3590628 prices: max = 124.580000 min = 49.870000 avg = 94.970000
2007: quan: total=3690568 prices: max = 82.820000 min = 50.640000 avg = 64.252500
2006: quan: total=3734226 prices: max = 66.130000 min = 51.800000 avg = 57.815000
2005: quan: total=3754671 prices: max = 57.420000 min = 35.270000 avg = 46.934167
2004: quan: total=3820979 prices: max = 41.840000 min = 28.570000 avg = 34.434167
2003: quan: total=3676005 prices: max = 30.390000 min = 24.050000 avg = 27.067500
2002: quan: total=3418022 prices: max = 26.250000 min = 16.290000 avg = 22.513333
2001: quan: total=3471067 prices: max = 23.850000 min = 15.460000 avg = 21.418333
2000: quan: total=3399240 prices: max = 29.030000 min = 23.310000 avg = 26.415833
1999: quan: total=3228092 prices: max = 22.790000 min = 9.210000 avg = 15.789167
1998: quan: total=3242712 prices: max = 14.310000 min = 9.380000 avg = 11.504167
1997: quan: total=3069431 prices: max = 21.860000 min = 16.050000 avg = 17.770000
1996: quan: total=2893646 prices: max = 21.630000 min = 16.100000 avg = 18.930833
1995: quan: total=2767313 prices: max = 17.400000 min = 15.060000 avg = 15.804167
1994: quan: total=2704197 prices: max = 16.090000 min = 11.640000 avg = 14.120000
1993: quan: total=2543375 prices: max = 16.710000 min = 12.200000 avg = 15.126667
1992: quan: total=2294570 prices: max = 18.270000 min = 14.420000 avg = 16.703333
1991: quan: total=2146066 prices: max = 22.980000 min = 16.150000 avg = 17.516667
1990: quan: total=2216604 prices: max = 30.080000 min = 0.090000 avg = 19.206667
1989: quan: total=1963917 prices: max = 17.970000 min = -0.340000 avg = 15.001667
1988: quan: total=1887860 prices: max = 22.400000 min = 11.520000 avg = 14.306667
1987: quan: total=1744978 prices: max = 18.070000 min = 13.860000 avg = 16.676667
1986: quan: total=1634789 prices: max = 25.800000 min = 10.090000 avg = 14.392500
1985: quan: total=1260740 prices: max = 27.380000 min = 25.420000 avg = 26.220000
1984: quan: total=1319723 prices: max = 28.050000 min = 27.300000 avg = 27.670833
1983: quan: total=1293817 prices: max = 32.500000 min = 28.590000 avg = 29.583333
1982: quan: total=1420754 prices: max = 35.530000 min = 32.250000 avg = 33.359167
1981: quan: total=1763072 prices: max = 36.920000 min = 33.650000 avg = 35.106667
1980: quan: total=1977246 prices: max = 33.940000 min = 26.320000 avg = 31.555000
```

```
1979: quan: total=2467315 prices: max = 24.570000 min = 13.640000 avg = 18.675833
1978: quan: total=2392351 prices: max = 13.590000 min = 13.360000 avg = 13.434167
1977: quan: total=2519806 prices: max = 13.620000 min = 12.540000 avg = 13.323333
1976: quan: total=2050424 prices: max = 12.590000 min = 12.230000 avg = 12.420000
1975: quan: total=1584730 prices: max = 12.110000 min = 11.310000 avg = 11.588333
1974: quan: total=1463864 prices: max = 12.090000 min = 6.710000 avg = 11.087500
1973: quan: total=1392970 prices: max = 5.270000 min = 2.730000 avg = 3.290833
```

12 Problem 5a - countBases

- define 'countBases(dna)' - returns the number of 'A', 'C', 'G', 'T' in a DNA strand

```
In [111]: bases = 'ACGT'
          dna = 'CATCGATATCTCTGAGTGCAC'
```

```
In [113]: countBases('AC')
```

```
Out[113]: [1, 1, 0, 0]
```

```
In [114]: countBases(dna)
```

```
Out[114]: [5, 6, 4, 6]
```

13 Problem 5b - reverseComplement

- define 'reverseComplement(dna)'
- swaps A <-> T, C <-> G, and returns the new DNA in reverse order

```
In [116]: reverseComplement('ACGT')
```

```
Out[116]: 'ACGT'
```

```
In [117]: reverseComplement(dna)
```

```
Out[117]: 'GTGCACTCAGAGATATCGATG'
```