Homework2

January 27, 2017

0.1 Introduction to Python - Homework #2

- Due Thursday Feb 2 at Noon
- This homework will be graded
- You MUST submit on Courseworks2 BEFORE it closes
  - Email submissions are not accepted

1 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.

2 Tips

3 Make SURE you are running Python 3.5, and not 2.7

```
In [49]: # the 3.5.2 is the version

         import sys
         sys.version

Out[49]: '3.5.2 |Anaconda custom (x86_64)| (default, Jul  2 2016, 17:52:12) \n[GCC
```

4 use string.format

```
In [50]: # instead of

         name = 'larry'
         cnt = 5
         lst = [1,5,6]
         'name is ' + name + ', count is ' + str(cnt) + ', list = ' + str(lst)

Out[50]: 'name is larry, count is 5, list = [1, 5, 6]'

In [51]: 'name is {}, count is {}, list = {}'.format(name, cnt, lst)

Out[51]: 'name is larry, count is 5, list = [1, 5, 6]'
```

5 sum

```
In [52]: # sum is a builtin function that will sum a list

         sum([35.3, 3,2])

Out[52]: 40.3
```

6 type

- can always find the type of an object with the 'type' function

```
In [53]: [type(234), type(range(2,44)), type(zip())]

Out[53]: [int, range, zip]
```

7 isinstance

```
In [54]: # 'isinstance' is a little more concise then 'type' in some situations

         type(234) == int
         isinstance(234, int)

Out[54]: True
```

8 zip

- zip stops after shortest list is exhausted

```
In [55]: list(zip(range(10), range(4, 7)))

Out[55]: [(0, 4), (1, 5), (2, 6)]

In [56]: # sum the integers in a nested list

         def rsum(x):
             if x == []:
                 return(0)
```

```
            if isinstance(x, list):
                return rsum(x[0]) + rsum(x[1:])
            else:
                # not a list
                return(x)

        rsum([3,4,[3,4,[4,2,3],3],3])
```

```
Out[56]: 29
```

9 these are all fairly short functions

- if you are doing something very complicated, think again, or get some help

10 Problem 1a

- write function 'dot'
- computes the standard 'dot products' between two lists
- example: dot([1,2,3], [4,5,6]) =

$$1 * 4 + 2 * 5 + 3 * 6 = 32$$

- if one vector is longer than the other, the extra elements are ignored

```
In [57]: # test vectors for Problem 1 a,b,c,d

         tv0 = [1,2,3]
         tv1 = [4,5,6,7,8,9]
```

```
In [59]: # the 7,8,9 elements are ignored because zip stops when the shorter list

         dot(tv0, tv1)
```

```
Out[59]: 32
```

11 Problem 1b

- write function 'shortlong'
- takes two vectors, and returns in a list the shorter vector, the short vector length, the long vector, and the long vector length

```
In [61]: shortlong(tv0, tv1)
```

```
Out[61]: [[1, 2, 3], 3, [4, 5, 6, 7, 8, 9], 6]
```

```
In [62]: shortlong(tv1, tv0)
```

```
Out[62]: [[1, 2, 3], 3, [4, 5, 6, 7, 8, 9], 6]
```

12 Problem 1c

- write function ‘dotmv’
- more flexible version of ‘dot’
- ‘dotmv’ takes an extra ‘offset’ arg, which moves the shorter vector to the right
- use ‘shortlong’
- dotmv(tv0, tv1, 2) =

$$1 * 6 + 2 * 7 + 3 * 8$$

```
In [64]: [dotmv(tv0, tv1, 0), dotmv(tv0, tv1, 1), dotmv(tv0, tv1, 2)]
```

```
Out[64]: [32, 38, 44]
```

13 Problem 1d

- write function ‘dotpad’
- another version of ‘dot’
- ‘dotpad’ takes a pad arg
- if one vector is shorter, it is padded on the right with the pad value
- use ‘shortlong’
- don’t modify the input vectors
- dotpad(tv0, tv1,1) = dot([1,2,3,1,1,1], [4,5,6,7,8,9])

```
In [66]: [dotpad(tv0, tv1, 0), dotpad(tv0, tv1, 1), dotpad(tv0, tv1,2)]
```

```
Out[66]: [32, 56, 80]
```

14 Problem 2

- write function ‘cbt’
- ‘cbt’ => ‘Collate By Type’
- argument: a non-nested list of objects
- returns: a dictionary, where there is a key for each type found. the value of each key is a list of the objects of that type found.
- prints:
- the number of each type found
- the sum, if any, of the ints, and floats found
- the strings, if any, sorted alphabetically, and concatenated togther, separated by ‘|’

```
In [2]: x = [23, 2**20, 3.14,'shapiro', 2**10+7, sorted,2.34, 'science', len, 43, '
        cbt(x)
```

```
found 4 of <class 'str'>
found 4 of <class 'int'>
found 2 of <class 'builtin_function_or_method'>
found 2 of <class 'float'>
sum of <class 'int'> is 1049673
sum of <class 'float'> is 5.48
alpha sorted concat of strings: butler|science|shapiro|unicode
```

```
Out[2]: {str: ['butler', 'science', 'shapiro', 'unicode'],
          int: [23, 43, 1031, 1048576],
          builtin_function_or_method: [<function sorted>, <function len>],
          float: [2.34, 3.14]}
```

15 Problem 3

- write function 'partition'
- divides a list into segments
- first arg is the input list
- second arg is the length of each segment. if there are not enough list elements to make a final segment of length n, they are discarded
- third arg is how many list elements should overlap btw adjacent segments
- remember range is range(inclusive, exclusive), range[0,2] => [0,1]
- might want to use 'while' instead of 'for'

```
In [73]: partition(list(range(10)), 2, 0)

Out[73]: [[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]]

In [74]: # only want length 3 partitions, so 9 was discarded

         partition(list(range(10)), 3, 0)

Out[74]: [[0, 1, 2], [3, 4, 5], [6, 7, 8]]

In [75]: partition(list(range(10)), 2, 1)

Out[75]: [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [8, 9]]

In [76]: partition(list(range(10)), 4, 0)

Out[76]: [[0, 1, 2, 3], [4, 5, 6, 7]]

In [77]: partition(list(range(10)), 4, 3)

Out[77]: [[0, 1, 2, 3],
          [1, 2, 3, 4],
          [2, 3, 4, 5],
          [3, 4, 5, 6],
          [4, 5, 6, 7],
          [5, 6, 7, 8],
          [6, 7, 8, 9]]
```

16 Problem 4a

- write function 'expandlazy'
- if given a 'lazy' range, zip, or enumerate, expand it into a list

```
In [80]: [expandlazy(234), expandlazy(range(3)), expandlazy('asdf'), expandlazy(enu

Out[80]: [234, [0, 1, 2], 'asdf', [(0, 'a'), (1, 'b'), (2, 'c')]]
```

17 Problem 4b

- write function 'expandlazylist'
- expand any lazy elements of a non nested list

```
In [82]: x = [1,2,3, range(4), 5, zip([1,2,3], [4,5]), 'asdf', enumerate(['a', 'b',
         expandlazylist(x)

Out[82]: [1,
          2,
          3,
          [0, 1, 2, 3],
          5,
          [(1, 4), (2, 5)],
          'asdf',
          [(0, 'a'), (1, 'b'), (2, 'c')]]
```

18 Problem 5

- 'flatten' turns a nested list into a non-nested linear one
- use recursion

```
In [84]: flatten([1,[2,3,4,[5,6,[7,8],9],11]])

Out[84]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 11]

In [85]: flatten([1,2,3,[4,56],[44,55],7,8])

Out[85]: [1, 2, 3, 4, 56, 44, 55, 7, 8]
```