Lab 4: Lists of lists

Feb 6, 2018

Main Event

1. If-statement practice

There will be more if-statement practice in the next lab, but for now, let’s try some basics:

- Ask the user for a number. Print whether that number is odd or even.
- Write a program that will comment on a person’s age. Have the user input their age. If they are between the ages of 0 and 20 (inclusive), print a message (“You’re young!”, for example); if they are older than 20, but less-than or equal-to 50, print something else; and if they are older than 50 print something else. If the age seems outrageous (less-than zero, or older than 120), tell the user they are out of range.

  Note that in python you can combine comparisons:

  ```
  if x < y and y < z:
    print('Hello')
  ```

  is the same as

  ```
  if x < y < z:
    print('Hello')
  ```

Solution:

```
# even/odd
num = int(input('Please enter a number: '))
if num % 2:
    print('odd')
```

```
else:
    print('even')

# age comments
age = int(input('Enter your age: '))

if 0 < age <= 120:
    if age > 50:
        print('Respect')
    else:
        if age > 20:
            print('Sweet!')
        else:
            print('Baby!')
else:
    print("I don't know that age")
```

2. Fun with string output

Given the following list of lists:

```
list_of_lists = [['a', 'b', 'c'], ['d', 'e', 'f'], ['g', 'h', 'i']]
```

1. Print the following output:

```
.a
...b
.....c
.......d
.........e
...........f
.............g
...............h
.................i
```

2. Now print the following output:

```
.a
.b
.c
...d
...e
```

```
...f
.....g
.....h
.....i
```

If you’re having trouble, count the number of dots prior to each letter and correlate that number to each characters position in the original list.

Solution:

```python
list_of_lists = [['a', 'b', 'c'], ['d', 'e', 'f'], ['g', 'h', 'i']]

# 1
dots = 1
for innerList in list_of_lists:
    for i in innerList:
        print('.' * dots + i)
        dots += 2

# 2
dots = 1
for innerList in list_of_lists:
    for i in innerList:
        print('.' * dots + i)
    dots += 2 # the only change!
```

3. Sublist features

Consider a list of lists:

```python
list_of_lists = [[ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ], [ 10, 11, 12 ]
```

1. Use a for-loop to count the total number of elements in the list. In this example, your programs output would be 12. Remember, you can use the function `len` to get the length of a sequence.

2. Create a new list containing the product of the respective sublists. Your final output should be

   ```
   [ 6 , 120, 504, 1320 ]
   ```

Solution:

```
list_of_lists = [[ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ], [ 10, 11, 12 ]

prods = []
for row in list_of_lists:
    r = 1
    for i in row:
        r *= i
    prods.append(r)
print(prods)
```

4. Random sublist features

Redo the previous exercise on a list you generate randomly!

To do this, first copy your code from the previous exercise into a new file. Update the code such that `list_of_lists` is not “static.” Specifically, that means instead of using the 4-list 3-element list that was provided (`list_of_lists`), generate a new list-of-lists that has *n* lists containing *m* elements; where *n* and *m* are generated randomly.

The values that go into the sublists can also be random. If you are tired of using random integers, consider using random floats:

```
random.uniform(x, y)
```

where `x` and `y` are range specifiers.

If you did the previous exercise correctly, your code for element counting and product aggregation should continue to work *without alteration*.

Solution:

```
import random

n = random.randrange(1, 20)
m = random.randrange(1, 20)

# create the list
for i in range(n):
    sublist = []
    for j in range(m):
        value = random.uniform(1, 2)
        sublist.append(value)
    list_of_lists.append(sublist)

# do the same thing as last time
prods = []
for row in list_of_lists:
    r = 1
    for i in row:
        r *= i
    prods.append(round(r, 2))
print(prods)
```

5. Random words

Create a list containing strings of arbitrary size. Doing this will require a loop to generate the strings, inside of a loop that builds the list.

To build strings, a loop must first decide how long the string will be — you can use a random value between three and eight. Next, within the loop, you should decide the value of a single character. You can do this using Python's chr function, which takes an integer and returns a string of length one (based on the ASCII value):

```
>>> chr(97)
'a'
```

Lower case letters between "a" and "z" have values 97 through 122 (97+25). Thus, you should use a for-loop to build a string character-by-character (we did this in the previous lab).

The outer loop should be used to store completed strings in a list. When the (outer) loop is finished, print the list of strings, with each string on a separate line.

Solution:

```
import random

words = 10
corpus = []
```

```
for w in range(words):
    string = ''
    n = random.randrange(3, 9)
    for i in range(n):
        c = random.randrange(97, 97 + 26)
        string += chr(c)
    corpus.append(string)

for i in corpus:
    print(i)
```

6. Pretty printing

Turn the following list of lists:

```
list_of_lists = [['a', 'b', 'c'], ['d', 'e', 'f'], ['g', 'h', 'i']]
```

into a nicely formatted box:

```
+-+-+-+
|a|b|c|
+-+-+-+
|d|e|f|
+-+-+-+
|g|h|i|
+-+-+-+
```

Specifically, decorate the contents of the list(s) with the formatting characters `+` and `-`.

One strategy to do this is to keep the formatting characters within the list of lists. That’s not a good idea. Instead use characteristics of the data structure to figure out how many decorators to use and how those decorators should be laid out. You’ll know you have done this correctly if the code you have written works for lists of different sizes without altering any of the code you’ve written for the original list. For example,

```
list_of_lists_A = [
   ['a', 'b', 'c', '1'],
   ['d', 'e', 'f', '2'],
   ['g', 'h', 'i', '3']
]
```

becomes

```
+-+-+-+-+
|a|b|c|1|
+-+-+-+-+
|d|e|f|2|
+-+-+-+-+
|g|h|i|3|
+-+-+-+-+
```

while

```
list_of_lists_B = [
   ['a', 'b', 'c', '1'],
   ['d', 'e', 'f', '2'],
   ['g', 'h', 'i', '3'],
   ['j', 'k', 'l', '4']
]
```

becomes

```
+-+-+-+-+
|a|b|c|1|
+-+-+-+-+
|d|e|f|2|
+-+-+-+-+
|g|h|i|3|
+-+-+-+-+
|j|k|l|4|
+-+-+-+-+
```

using the same code.

Solution:

```
l = [['a', 'b', 'c'],
     ['d', 'e', 'f'],
     ['g', 'h', 'i']]

# l = [['a', 'b', 'c', '1'],
#      ['d', 'e', 'f', '2'],
#      ['g', 'h', 'i', '3']]
# l = [['a', 'b', 'c', '1'],
#      ['d', 'e', 'f', '2'],
#      ['g', 'h', 'i', '3'],
#      ['j', 'k', 'l', '4']]

row_length = len(l[0]) # assume every row has the same length
separator = '+-' * row_length + '+'

print(separator)
for sublist in l:
    row = '|'
    for i in sublist:
        row += i + '|'
    print(row)
    print(separator)

# There are numerous ways to do this! The proposed solution to "Making
# the band" presents another.
```

Additional Practice

1. Making the band

Create a 3-by-3 grid:

```
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
```

Put x's in the cube based on user specification. Specifically, the user will provide two sets of coordinates; your job is to make a continuous line of x's in the cube based on those coordinates.

```
Please enter the first coordinate: 0,0
Please enter the second coordinate: 0,2
```

This would place a band of `x`'s down the first (0th) column:

```
+-+-+-+
|x| | |
+-+-+-+
|x| | |
+-+-+-+
|x| | |
+-+-+-+
```

You can assume that in the input given by the user either the columns will be the same or the rows will be the same. For example, if the first coordinate entered were *x*,*y*, and the second coordinate *a*,*b*, either *x* would equal *a*, or *y* would equal *b*.

Solution:

```
first = input('Please enter the first coordinate: ').split(',')
second = input('Please enter the second coordinate: ').split(',')

# by default, input returns a string
col_0 = int(first[0])
row_0 = int(first[1])
col_1 = int(second[0])
row_1 = int(second[1])

# we have a 3x3 board
board_rows = 3
board_cols = board_rows

#
# Approach 1: First create the board...
#
cube = []
for i in range(board_rows):
    line = []
    for j in range(board_cols):
        line.append(' ')
    cube.append(line)

# ... then populate the contents
for i in range(row_0, row_1 + 1):
    for j in range(col_0, col_1 + 1):
        cube[i][j] = 'x'

notch = '+' + '-+' * board_cols
print(notch)
for row in cube:
    print('|', '|'.join(row), '|', sep='')
    print(notch)

#
# Approach 2: Create the board and populate the contents in the same
# step
#
cube = []
for i in range(board_rows):
    line = []
    row = row_0 <= i <= row_1
    for j in range(board_cols):
        col = col_0 <= j <= col_1
        if row and col:
            line.append('x')
        else:
            line.append(' ')
        # or just line.append('x' if row and col else ' ')
    cube.append(line)

notch = '+' + '-+' * board_cols
print(notch)
for row in cube:
    print('|', '|'.join(row), '|', sep='')
    print(notch)
```

2. Matrix transposition

A list of lists can also be thought of as a matrix. In this context, sublists can either be row values or column values, depending on your interpretation (known as row-major and column-major, respectively).

Consider the matrix

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

Assuming this is row-major, it would be pretty printed (see previous) as follows:

```
+-+-+-+
|1|2|3|
+-+-+-+
|4|5|6|
+-+-+-+
|7|8|9|
+-+-+-+
```

The objective of this exercise is to transpose that matrix and pretty print the results. Specifically, the rows should become columns, and the columns should become rows:

```
+-+-+-+
|1|4|7|
+-+-+-+
|2|5|8|
+-+-+-+
|3|6|9|
+-+-+-+
```

Solution:

```
# create the matrix
n = 2
m = n
val = 1

original = []
for i in range(n):
    l = []
    for j in range(m):
        l.append(val)
        val += 1
    original.append(l)

# print the original
print('Original')
for row in original:
    innerList = []
    for cell in row:
        innerList.append(str(cell))
    print(' '.join(innerList))

# make a "deep" copy of original
swap = []
for i in original:
    swap.append(i.copy())

# transpose!
for i in range(len(original)):
    row = original[i]
    for j in range(len(row)):
        swap[j][i] = original[i][j]

# print the transposition
print('Transposed')
for row in swap:
    innerList = []
    for cell in row:
        innerList.append(str(cell))
    print(' '.join(innerList))
```