Lab 3: For loops

Feb 1, 2018

Main Event

1. Integers

Addition

Write a for-loop to add the numbers 1–10. Once the loop is finished, print the value.

Multiplication

Write another for-loop that multiplies the numbers 1 through 100. Remember, the `range` function is your friend.

Solution:

```
# addition
total = 0
for i in range(11):
    total += i
print(total)

# multiplication
total = 1
for i in range(1, 100 + 1):
    total *= i
print(total)
```

2. Strings

Consider the following string:

```
s = 'The rain in Spain falls mainly on the plain'
```

Complete the following tasks only using for-loops over `s`. While it may be tempting to use string methods or Python built-in functions to accomplish these tasks, don't! Each can be performed without altering the original string. Specifically you should either looping over the original string:

```
for i in s:
```

or the original strings' indices:

```
for i in range(len(s)):
```

Good luck!

Length

Use a for-loop to count the number of characters in `s`.

Create

Use a for-loop to generate a new string `s1`, where `s1` is an exact replica of `s`.

Reverse

Use a for-loop to generate a new string `t`, where `t` is the reverse of `s`. After the loop is finished, print `t` to verify your work.

Again, you cannot use any built-in string methods, on either `s` or `t`, nor do you need to convert `s` to a list. Remember, *string addition is not commutative*!

Upper

Use a for-loop to generate a new string `v`, where `v` is the upper-case version of `s`. It is okay if you call `upper` on the individual character.

Count

Count is a useful string method. Here we will implement it ourselves! Consider a new string

```
c = 'a'
```

Use a for-loop to count the number of occurrences of `c` in `s`. As with previous assignments, we do not have to use an if-statement here; it is enough to cast the Boolean equality test to an integer:

```
>>> s1 = 'a'
>>> s2 = 'b'
>>> int(s1 == s2)
0
```

If you do not understand what's going on here, change the value of `s2` to 'a' and perform the cast again.

Generalized count

Count the number of occurrences of `c2` in `s`, where

```
c2 = 'ain'
```

Whereas the last exercise compared a character to each character of `s`, this exercise can be thought of as comparing a string to a *slice* of `s`.

Solution:

```
s = 'The rain in Spain falls mainly on the plain'

# length
length = 0
for i in s:
    length += 1
print(length)

# create
s1 = ''
for i in s:
    s1 += i
print(s1)

# reverse
t = ''
for i in s:
    t = i + t
print(t)
```

```
# count
c = 'a'
total = 0
for i in s:
    total += int(c == i)
print(total)

# generalized count
c = 'ain'
total = 0
for i in range(len(s) - len(c) + 1): # range(len(s)) is also fine
    window = s[i:i + len(c)]
    total += int(window == c)
print(total)
```

3. Lists

Summation

Create a list of integers. Produce a new list that is the cumulative sum of that list:

```
>>> original = [ 1, 2, 3, 4 ]
>>> # ... you create sumlist
>>> print(sumlist)
[ 1, 3, 6, 10 ]
```

It's probably best to develop this in stages: using a list of three or four numbers to ensure your *algorithm* is correct, then move on to larger list sizes to really see it in action.

Randomization!

1. Use Python's random number generator to create a list of random length containing values that are also random. You should be familiar with random numbers in Python. This time we will use a different function from the random library:

   ```
   import random
   random.randrange(10)
   ```

produces a random number between 0 and 9. Like the `range` function, you can give two arguments to start from something other than zero:

```
random.randrange(5, 10)
```

The range you use is up to you, but should probably start at one (why?).

2. Once that is complete, use a separate for-loop to find the sum of your new list. The final output of this program should be the list size, followed by its summation.

In developing your program, it is advisable to first use relatively small ranges for the random number

```
random.randrange(1, 5)
```

and to print the generated list along with the final output. Once you are confident your code is correct, comment out the statement that prints the list, and increase the possible range of list sizes:

```
random.randrange(10 ** 3, 10 ** 4)
```

Solution:

```
import random

original = [ 1, 2, 3, 4 ]

# summation
total = 0
sumlist = []
for i in original:
    total += i
    sumlist.append(total)
print(sumlist)

# random creation...
random_list = []
size = random.randrange(10 ** 3, 10 ** 4)
for i in range(size):
    value = random.randrange(1, 100)
    random_list.append(value)
```

```
# print(random_list)

# ... and summation
total = 0
for i in random_list:
    total += i
print(total)
```

4. Loop transformation

Recall our exercise on counting vowels. At the time, we had five separate calls to count for each vowel:

```
myString = 'The quiCk brOwn fOx jumped over thE lAzy dog'
aCount = myString.count('a')
eCount = myString.count('e')
iCount = myString.count('i')
oCount = myString.count('o')
uCount = myString.count('u')
print(aCount + eCount + iCount + oCount + uCount)
```

Rewrite this code using a for-loop so that it is more succinct. If you don't know where to begin, think about the following:

1. What statements are being repeated?
2. What is changing in those statements and how can that change be captured in a sequence?
3. How can the techniques from previous exercises keep a tally of what's going on?

You should count all vowels, regardless of casing.

Solution:

```
myString = 'The quiCk brOwn fOx jumped over thE lAzy dog'
count = 0
for i in 'aeiou':
    count += myString.lower().count(i)
print(count)
```

Additional Practice

1. Pyramid

Ask the user for an integer. Use a single for-loop to produce equilateral triangle of stars, where the triangle's edge length is the specified integer. As an example:

```
>>> Please enter an edge length: 5
    *
   * *
  * * *
 * * * *
* * * * *
```

Solution:

```
width = int(input('Please enter an edge length: '))

for i in range(1, width + 1):
    print(' ' * (width - i) + '* ' * i)
```

2. String rotation

String rotation is the act of shifting letters in a string. As an example: the letters on a compass are often printed in a circle, like a clock with four numbers. If you were to lay the letters in a straight line, assuming north were the first letter (since it's on top), you'd have: NESW. If we were to right-rotate this string by one, we would have WNES (and subsequently, if we were to put it back on the compass, west would be on top).

1. Create a for-loop that rotates an arbitrary string by one place. It is okay if you use the compass example from the explanation.
2. Do the same thing using string slicing.
3. Which is faster? Here, we'll look at a rudimentary way to time a *block* of code using timeit module:

   ```
   import timeit
   start_time = timeit.default_timer()
   # your code block
   ```

```
print(timeit.default_timer() - start_time)
```

A couple of things to note:

1. Your “code block” in the for-loop case is the for-loop itself; in the slice case it’s the one (or more) lines you’ve used to do the slicing—do not include any code that creates the original string in this block.
2. You might not get interesting results if your string is too short. Use another for-loop to create a really large string (~2^{20} characters work decently on my machine). You can use the `random` function from above to generate characters (just cast the returned integer to a string).

Finally, run it a few times to give some confidence to your conclusions. If you’re really particular, put the two (slicing and looping) in separate files to ensure caching or optimization isn’t coming into play. What does this say about the inner workings of Python?

4. Redo this exercise such that the rotation is generic. Here, we’ve looked at a single rotation; make an update so that you can do any number of rotations. For example, rotate the compass by three: WSEN.

Solution:

```
import random
import timeit

# first, create a random string.
rubbish = ''
for i in range(2 ** 21):
    rubbish += str(random.randrange(10))

# test the for loop version
rotation_a = ''

start = timeit.default_timer()
for i in rubbish[1:]:
    rotation_a += i
rotation_a += rubbish[0]
stop = timeit.default_timer()

print(stop - start)
```

```
# test the native slice version
rotation_b = ''

start = timeit.default_timer()
rotation_b = rubbish[1:] + rubbish[0]
stop = timeit.default_timer()

print(stop - start)
```

3. Higher order functions

For-loops rely on the ability to continuously *bind* a value to a *free* variable. Consider the statement

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

where `x` is some sequence. `i` is initially free, but will be bound to values in `x` as the loop progresses. In some programming languages, once a variable is bound, it can no longer be bound again. Such languages thus do not have loops. To perform iteration, they instead must rely on applying a function to pieces of a sequence.

Python has various methods that emulate this behavior; most notably map and reduce. Both are functions that take two parameters: a function, and a sequence. They subsequently apply the given function to the sequence: `map` one element at a time; `reduce` two elements at a time (sort of: it's actually the first two elements the first time, and the result of the function over those two elements with individual remaining elements subsequent times). The e-book "Python Tips" contains good documentation of the basics of these methods.

After reading this documentation, challenge yourself to do some of the previous exercises using `map` or `reduce`:

Integers

Addition and multiplication can be accomplished using `reduce`.

Strings

Reverse can be accomplished using `reduce`. Length, single character count, and upper can be accomplished using `map`. In the case of count, the Python built-in

function sum, which adds a list of integers, may help. In the case of upper, you can pass the result of `map` to the string method join to produce the final string.

Lists

Summation, and randomization can be accomplished using `reduce` and `map`, respectively. In the documentation for `reduce`, in fact, the example given is for summation:

```
>>> from functools import reduce
>>> help(reduce)
```

Solution:

```
from random import randrange
from functools import reduce

# Integers
print('Sum of 1 to 10', reduce(lambda x, y: x + y, range(1, 11)))
print('Product of 1 to 100', reduce(lambda x, y: x * y, range(1, 100 +

# Strings
s = 'The rain in Spain falls mainly on the plain'

print(s)
print('\t', 'reversed:', reduce(lambda x, y: y + x, s))
print('\t', 'characters', sum(map(len, s)))
print('\t', "a's", sum(map(lambda x: x == 'a', s)))
print('\t', 'uppercased', ''.join(map(lambda x: x.upper(), s)))

# Lists
print('summation', reduce(lambda x, y: x + y, [ 1, 2, 3, 4 ]))

l = map(lambda x: randrange(100), range(randrange(1, 11)))
print('random list', list(l))
```

Introduction to Computer Science
jerome.white@nyu.edu

jerome-white

Learning computer science concepts through practice.