Lab 2: Sequences

Jan 30, 2018

Main Event

1. Pony Stash Token

Define `x` as follows:

```
x = 'pony stash token'
```

Use indexing, slicing, and stepping to achieve the following:

1. Print the first letter: `p`
2. Print the last letter: `n`
3. Print all letters except the first and the last: `ony stash toke`
4. Print every other character: `pn ts oe`
5. Produce the string `python`
6. Print the last word (of `x`) in reverse. While this question, like the previous, can be done with a single subscript notation, doing so is not straightforward. It is okay in this case use a multi-step process—using slicing more than once—to accomplish the task.
7. Update `x` such that “stash” is replaced with “tony”. While it may be tempting to use replace, don’t. (Again, it is okay to do this in multiple stages.)

These can all be accomplished without converting the string to a list!

Solution:

```
x = 'pony stash token'
```

```
print(x[0],      # first letter
      x[-1],     # last letter
      x[1:-1],   # except first and last
      x[::2],    # every other
      x[::3],    # "python"
      sep='\n')

# There are several ways to accomplish the "last backwards". Here are
# two:
print(x[-len('token'):][::-1],
      x[:10:-1],
      sep='\n')

x = x[:5] + 'tony' + x[10:]
print(x)
```

2. Computer names

Computer user names are often some combination of a persons actual name. For example, NYU uses your initials—the respective first letters of your family and given names—along with a number. Write a program that asks for a persons name, then prints their computer name. In this case, their computer name should be the first letter of their given name, and the first five letters of their family name in reverse, all in lower case. For example:

```
Please enter your full name: Bob Hopefully
Your computer name is: bfepoh
```

For this exercise, assume that the final name provided is the family name. Thus, if the user were to have entered “Bob Paul Mitchell Hopefully”, the computer name would have been the same (`bfepoh`).

Solution:

```
name = input('Please enter your full name: ')

first_initial = name[0]

names = name.split()
last_name = names[-1]
last_five = last_name[4::-1] # last_name[:5][::-1] is also fine
```

```
complete = first_initial + last_five

print(complete.lower())

# Or, in a single line:
# print((name[0] + name.split()[-1][4::-1]).lower())
```

3. List interaction

This exercise will have you manipulate a list based on interaction with the user. List method documentation can be found on the Python website.

1. Ask the user to provide a list of words, separated by commas. Print the sorted version of their input:

   ```
   >>> Please enter a list of words separated by commas: mother,dog,h
   ['dog', 'honey', 'mother']
   ```

2. Ask the user for another set of names, again separated by commas. Add these values to the previous list, sort, and print:

   ```
   >>> Please enter another list of words separated by commas: red,ap
   ['apple', 'dog', 'honey', 'mother', 'red']
   ```

   It may be enticing to use `append`, but definitely check out `extend`.

3. Ask the user for another word, and print the index of that word:

   ```
   >>> Enter a word: mother
   3
   ```

   Do not worry about whether the word is in the list. You can assume that the user will never make a mistake, and that the word they enter will have been previously added.

4. After printing the index of the word, remove the word from the list and re-print:

   ```
   >>> Enter a word: mother
   3
   ['apple', 'dog', 'honey', 'red']
   ```

Solution:

```
# 1
x = input('Please enter a list of words separated by commas: ')
y = x.split(',')
y.sort()
print(y)

# 2
x = input('Please enter another list of words separated by commas: ')
y.extend(x.split(','))
y.sort()
print(y)

# 3
x = input('Please enter a word: ')
index = y.index(x)
print(index)

# 4
y.pop(index)
print(y)
```

4. Dice game

"Dice game" is the second most amazing one-player game ever created: you first guess a number, you then roll a die to see if the die was able to guess your number. Let's write a program that does this; it should do four things:

1. **Roll the dice:** We can use Python's random library to "roll":

   ```
   import random
   random.randint(x, y)
   ```

   where `x` and `y` are range specifiers; from the documentation, `random.randint(x, y)` will

   > *return a random integer N such that x is less-than or equal-to N, and N is less-than or equal-to y.*

   Play with the above statements to see what you get. Remember, you can store the return value of `random.randint` into a variable for later use.

2. **Ask the user for a number:** This should be straightforward—we did this in the last lab.

3. **Print the output of the die:** The output should be printed in English, not numerically! That is, if 6 were rolled, the output should be the word “six”. It may be tempting to create this output using a loop (if you’re familiar with that concept). Don’t. This can be done by creatively using the data structure discussed in today’s class.

4. **Tell the user if they were right or wrong:** If the die correctly guessed the users value, congratulate the user! Again, it may be tempting to write this using an if-statement. Again, don’t. Instead, use data structures discussed in today’s class: we’ll talk about equality (and more generally Boolean) operators next week, but for now, suffice it to say that `==` tests for the equality of two variables. Further, you can print the result of the operation as though it were any other value—try it:

   ```
   >>> print(1 == 1)
   True
   >>> print(1 == 0)
   False
   ```

   Note that the equality expression can be cast to produce an integer:

   ```
   >>> print(int(1 == 1))
   1
   >>> print(int(1 == 2))
   0
   ```

   Thus, you can create such decision-like output using the structure(s) we talked about today in class—give it some thought!

If you’re having trouble, think about how the types discussed in today’s class can be used to move between integers (that you’re given or know how to produce) and something else that you might want (strings and sentences).

Solution:

```
import random

die = ['one', 'two', 'three', 'four', 'five', 'six']
ans = ['Incorrect', 'Correct']
```

```
roll = random.randint(1, len(die))

guess = int(input('Enter a number: '))
word = die[roll]
response = ans[int(guess == roll)] + '!'

print(response)
```

Additional Practice

1. Fancy dice

In addition to displaying the English version of the die, do so making it actually look like a die. Specifically, put it in a box that's programatically formatted. For example, if 6 were rolled, you would print:

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

It may be tempting to create this output using a loop (if you're familiar with that concept). Don't! Again, we will talk about those later as well. The trick is to remember the concepts of string concatenation and repetition that we talked about previously—that's all you need!

Solution:

```
import random

die = ['one', 'two', 'three', 'four', 'five', 'six']
ans = ['Incorrect', 'Correct']

roll = random.randint(1, len(die))

guess = int(input('Enter a number: '))
word = die[roll]
response = ans[int(guess == roll)] + '!'

edge = '+' + '-' * (len(word) + 2) + '+'
```

```
print(edge, '| ' + word + ' |', edge, sep='\n')
print(response)
```

Introduction to Computer Science

Introduction to Computer Science

jerome.white@nyu.edu

jerome-white

Learning computer science concepts through practice.