420-LCU-05 Programming in Python - Solution: Lab Exercise 5

March 21, 2022

Goals for this lab:

- Practice with lists, including:
  - `for` loops
  - `split()` and `join()`

Part 1 - List Basics

1. Create a variable named `ls1` that refers to a list of any five integers of your choice.

```
      ls1 = [5, 9, 11, 13, 17, 22]
```

2. Create another variable named `ls2` that refers to a list of five more integers (be sure it is not the exact same as `ls1`).

```
      ls2 = [6, 8, 10, 12, 13, 5]
```

3. Check whether Python considers your `ls1` to be greater than `ls2`.

```
if (ls1 > ls2):
    print("ls1 is greater than ls2")
elif (ls2 > ls1) :
    print("ls2 is greater than ls1")
#  prints "ls2 is greater than ls1" because 6 > 5
```

4. Write the statement to subtract 2 from the final element of `ls1`.

```
ls1[-1] -= 2 #or ls1[len(ls1)-1] -=2
print("ls1=",ls1)
```

5. Write the statement to combine `ls1` and `ls2` into a single new list `ls3`.

```
ls3 = ls1 + ls2
print("ls3=",ls3)
```

6. Append a value of your choosing to the list `ls1`. Check whether this affects the value of `ls3`.

```
ls1.append(19)
print(ls3) # should be unchanged.
```

7. Now make `ls4` an alias for `ls2` by writing

```
ls4 = ls2
print("ls4=",ls4)
print("ls2=",ls2)
```

8. Append something to `ls2` and check the new value of `ls4`.

```
ls2.append(18)
print("ls2=",ls2)
print("ls4=",ls4)# Change should affect ls2 and ls4 identically.
#ls2 and ls4 are aliases or 2 names for the same variable in memory.
```

Part 2 - List Methods

1. Create a variable named `ID` whose value is a string that represents your ID number. For example, if your ID is 1234567, you would write:

   ```
   ID = "1234567"
   ```

2. Create a variable named `ID_digits` whose value will be a list consisting of all of the digits in your ID number. Hint: Use the `list()` function to accomplish this.

   ```
   ID_digits = list(ID)
   print("ID=",ID)
   print("ID_digits=",ID_digits)
   ```

3. Write the statement or expression to count all of the occurrences of the digit '1' in `digits`.

   ```
   print("There are",digits.count('1'),"occurrences of the digit '1' in my ID")
   ```

4. Write the statement or expression to add the digit '9' to the **beginning** of the list.

   ```
   # Either of these two will work:
   ID_digits = ['9'] + ID_digits
   ID_digits.insert(0, '9')
   ```

5. Write the statement or expression to combine all of the elements of the list back into a single string consisting just of the characters in the list (no separators). **Hint:** use `join()`.

   ```
   new_string = ''.join(ID_digits)
   ```

Part 3 - introducing the `for` loop

One of the most basic aspects of programming is *iteration*. Iteration is just the repetition of a computation over some sequence of values. Most programming languages express iteration using one or more *loop* statements. There are two different loop statements in Python, `while` and `for`. Here, we will introduce the `for` loop. We will use a few examples to show how a `for` loop can be used to access each of the elements of a string in order. The `for` loop works by repeating one or more statements, called the loop *body*, for each element in a sequence variable such as a string. The syntax is as follows:

```
for x in sequence :
   statement1
   statement2
   ...
```

The variable `x` (you can use any name you like here) will be assigned the value of the first element (character) in the sequence, and the list of statements will be executed. Then the next element of the sequence will be assigned to `x`, and the list of statements will be executed again. This process continues until the last element of the sequence is used.

For this part and next, you can continue in the same python file and type the following code:

1. As in previous lab, define the variable `full_name` with the value of your full name.

   ```
   for letter in full_name:
       print(letter)
       print(letter.lower())
   ```

   In a comment describe what both statements inside the `for` loop do.

   ```
   # print each letter in my name as is. Then print each letter in lower case.
   ```

2. Now we'll do something a bit more interesting. Type in the following:

```
result = 0
for letter in full_name:
    print(letter, result)
    result += 1
print(full_name, result)
```

In a comment, describe what is printed, and why?

```
# For each iteration, it prints each letter in my name and the counter.
# Then prints full name and the total count of letters.
```

3. Here is our final example. Type in the following:

```
result = ''
for letter in full_name:
    result = letter + result

print(result)
```

In a comment, describe what is printed, and why.

```
# print full name backwards
# because at each iteration, a letter is read from the name and placed at the beginning of result.
```

Part 4 - More `for` loops

1. Assuming you still have the variable `ID_digits` defined, type

```
for digit in ID_digits:
    print(digit, int(digit) % 2 != 0)
```

Briefly explain what this statement does.

This loop prints one line for each digit in the list `ID_digits`. Each line consists of the digit and a Boolean indicating whether the value of the digit as an integer is odd (prints True) or even (prints False).

2. Now try the following. It will do something vaguely familiar:

```
result = []
for digit in ID_digits:
    result = [digit] + result
print(result)
```

Again, observe and briefly describe what this does. In particular, are the brackets necessary in the expression `[digit] + result`? Can you leave them out?

This loop will produce a list that is the *reverse* of the elements in the original list. Again, this is because successive elements of the original list is added to the beginning of the new list. The brackets are needed because list addition requires that both operands be of type `list`.

3. Now combine the elements of `result` back into a single string:

```
id_backward = ''.join(result)
print(id_backward, 'x'.join(result))
```

Be sure to show what is printed out.

```
# In this case, the code should print
76543219 7x6x5x4x3x2x1x9
```