Session 11: More Sequences

Name: _______ **SOLUTION** _________ CM: _________ Section: _____ Grade: ________ of 10

1. Show the output of these expressions:

   `print(3 + 3)` ___6___ **`print ("3" + "3")`** ___33___

   Why are the outputs different? The first is the ***addition operation*** as in ordinary arithmetic, while the second is what is called ***string concatenation***.

2. What is the output of the code shown to the right?

```
nums = []
for k in range(5):
    nums = nums + [k * 2]
print(nums)
```

   `[0, 2, 4, 6, 8]`

3. Suppose that we modified the code in the preceding problem by replacing the **`nums = []`** line with **`nums = 0`** and dropping the **`[]`** surrounding **`k * 2`**, so that the code becomes like that shown to the right.

```
nums = 0
for k in range(5):
    nums = nums + k * 2
print(nums)
```

   a. What is the output of the modified code? ___20___

   b. The name (variable) ***nums*** is now badly chosen. What would be a better name for it?

   total or ***sum***

4. What happens in problem 2 if we forget the **`nums = []`** line altogether? Be specific.

   When the statement `nums = nums + k * 2` is encountered the first time through the loop, an exception is generated (i.e., the program crashes) because ***nums*** on the ***right***-hand-side is undefined (at this point of the run).

5. Suppose that we modified the code in the preceding problem yet again, so that it now looks like the code shown to the right.

```
nums = ""
for k in range(5):
    nums = nums + str(k * 2)
print(nums)
```

   a. What is the output of the modified code?

   `02468`

   b. What would go wrong if we omitted the **`str`** function call?

   When the statement `nums = nums + k * 2` is encountered the first time through the loop, an exception is generated (i.e., the program crashes) because ***nums*** is a string (namely, the empty string at that point of the run) and `k * 2` is an integer (namely, 0) and Python does not allow one to add an integer to a string. (continues on the back of this page)

6. Suppose that ***seq_of_seqs*** is a sequence of sequences, for example,

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

Write code that would print the ***length*** of each inner sequence, each on its own line (so the above example would print **3 2 1 3 0** but each on its own line).

```
for k in range(len(seq_of_seqs)):
    print(len(seq_of_seqs[k]))
```

7. Repeat the previous problem, but now looping BACKWARDS from the ***last*** element in ***seq_of_seqs*** to the ***first*** element (so the above example would print **0 3 1 2 3** but each on its own line).

Many solutions are possible, including the following:

`for k in range(len(seq_of_seqs) - 1, -1, -1):`
    `print(len(seq_of_seq[k]))`
	`last = len(seq_of_seqs) - 1`
	`for k in range(len(seq_of_seqs)):`
	    `item = seq_of_seq[last - k]`
	    `print(len(item))`

`for k in range(len(seq_of_seqs) - 1, -1, -1):`
    `item = seq_of_seq[k]`
    `print(len(item))`
	`index = len(seq_of_seqs) - 1`
	`for k in range(len(seq_of_seqs)):`
	    `item = seq_of_seq[index]`
	    `print(len(item))`
	    `index = index - 1`

8. The function shown to the right is intended to return **True** if the given sequence of numbers contains a negative number, and **False** otherwise. For example:

has_negative([5, 3, -4, 8]) should return **True**

has_negative([5, 3, 4, 8]) should return **False**

```
def has_negative(numbers):
    for k in range(len(numbers)):
        if numbers[k] < 0:
            return True
        ~~else:~~
            ~~return False~~
    return False
```

a. What does ***has_negative***, as written, in fact return when the argument is **[5, 3, -4, 8]**? **False**

b. Mark up the code to indicate the changes needed to make the code correct. No ***else*** clause, and the ***return*** is unindented to match the indentation of ***for***.

9. The function shown to the right is intended to return **True** if the given sequence of numbers is a *decreasing* sequence, that is, if each number in the sequence is less than or equal to the *next* number in the sequence. For example:

is_decreasing([15, 11, 4, 4, 1])
should return **True**

is_decreasing([15, 11, 4, 8, 1])
should return **False**
(since 8 is bigger than 4, its predecessor in the sequence).

```
def is_decreasing(numbers):
    for k in range(len(numbers) - 1):
        if numbers[k + 1] > numbers[k]:
            return False
    return True
```

a. Fill in the blanks with ***True*** and ***False*** in the appropriate places.

b. The function has a small error in the FOR statement. Mark up the code to correct the error.