CS 120: Intro to Computer Programming II

In-Class Activity - 01 Python Review - Day 1

Attribution:
Some of the activities may have been copied, or heavily inspired, by things I found on the web. After the activity is due, you may, if you are interested, ask for the sources.

WEIRDNESS
For almost all of the ICAs this semester, you will work with your group to come up with some answers (often code), and then submit a PDF to GradeScope; your TA will grade it by hand.

But for the first 3 activities (the ones associated with the Python Review section of class), we will do something different. In these, you will submit code to GradeScope to be **automatically graded.** Your TA won't be involved at all; it will be entirely automated.

The purpose of these activities is to get you familiar with GradeScope, before you need to use it for a Project. But especially, these exist to **teach you about testcases.** For each program that you write, I will only provide a **partial explanation.** You will need to download this ICA's zipfile, open up the various files inside it, and use those to better understand what GradeScope is expecting.

All of the files in the zipfile are text files, although they have funny names. You should be able to open all of them in your favorite text editor (or, if you prefer, in Visual Studio). Here's what you can expect:

- `.out`

  `.out` files give the expected output from a program or a testcase. Pay close attention to capitalization, puncuation, blank lines, and spaces.

  Every `.out` file is associated with exactly one input file or testcase.

- `.stdin`

  `.stdin` ("dot standard in") files represent something that should be typed by the keyboard, as the input to the program. Either type it directly into your program (being careful to type **exactly** what's in the file), or just cut-n-paste into your program.

- `.py`

  This is a program which runs, which will test code that you have provided. This is common if I've asked you to define a function; I will provide a program which imports that code, and then calls the function.

  To use this type of testcase, simply run the `.py` file. (It will need to be in the same directory as your code.)

- `.in`

  Sometimes, we write programs that will read from input files on disk. To make testing simple, what I normally do is that I ask you to read the file **name** from the keyboard, and then open it. You then read the file contents, and do something.

  To use this type of testcase, place the file into the same directory as your program. Then run your program normally. When your program asks for a filename, simply type the name of the file.

(activities begin on the next page)

Activity 1 - Turn in this one

Write a file named `ica_01_1_act1.py`

Read one line of input from the user, and do something with it. Read the testcases to figure out what.

Solution:

```
text = input()
print(text)
print(text)
```

Activity 2 - Turn in this one

Write a file named `ica_01_1_act2.py`

Read two lines of input from the user, and do something with them. Read the testcases to figure out what.

Solution:

```
data  = input().strip()
count = int(input())
for i in range(count):
    print(data)
```

Activity 3 - Turn in this one

Write a file named `ica_01_1_act3.py`

Write a function `str_mult(a,b)` which takes two parameters. The first is a string, and the second is an integer. Write a `for` loop (do **not** use multiplication!), and concatenate that many copies of the string to itself.

Return the resulting string.

Should you print anything out in this function? Check the testcases to find out.

Solution:

```
def str_mult(msg, count):
    print("The function str_mult() has been called.  For this activity, we won't print out any c
    retval = ""
    for i in range(count):
        retval += msg
    return retval
```

Challenge Activity - Do not turn in this one

A Challenge Activity is something a little more difficult than the rest. If you don't have time to get to it, no problem. But if you group finishes the required activities early, see if you can do the Challenge Activity as well.

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

Write a function which takes one parameter, a string. Print out, on your first line of output, the first three characters of the string. (You may assume that the string has at least 3.) Print out, on your second line, the **last** three characters. Then, if the string was so short that the first 3 and last three overlapped, print "Overlap" on a third line.

Solution:

```
def string_manipulation(msg):
    print(msg[:3])
    print(msg[-3:])
    if len(msg) < 6:
        print("Overlap")
```

(activity continues on next page)

Challenge Activity - Do not turn in this one

NOTE: For this one, turn in **both** an explanation, and also code which will demonstrate what you've discovered.

Write some code to investigate what happens if you try a slice which has "weird" start and stop points. Answer all of the following questions:

- What happens if the stop point of the slice (that is, the 2nd index) is a long way past the end of the string?
- What happens if the start and stop points are the same?
- What happens if the start and stop points are both positive, but the start is greater than the end?
- What happens if the start is beyond the end of the string?

Solution:

- If the stop point of the slice is past the end of the string, then Python will simply end the slice at the end of the string.
- The length of the returned string will be zero.
- If the start comes *after* the stop, then the slice will be empty.
- If the start is after the end of the string, then the slice will be empty.

Challenge Activity - Do not turn in this one

Write a function which takes a single parameter, an integer. Print out a square of asterisks on the screen, that many rows high and that many columns wide. For instance, if you were passed 5, you should print:

```
*****
*****
*****
*****
*****
```

Solution:

```
def square(wid):
    for i in range(wid):
        print("*" * wid)
```