CS 120 (Fall 21): Introduction to Computer Programming II

# Short Project #2

due at 5pm, Thu 2 Sep 2021

## 1 Overview

In this assignment, you will write several short Python programs, each doing a relatively small task.

**Precision**
Follow the directions given *exactly:* your code will be graded automatically, so any deviation from the program spec can result in a significant loss of credit. If you are unsure about something, ask for clarification in Office Hours or Piazza.

**Style**
Please pay attention to the programming style guidelines for this class. For this assignment you will be notified of style violations but not penalized for them; style violations will be penalized in subsequent assignments.

**Submitting Your Solution**
This assignment requires that you submit multiple files. Because of the the way that the auto-grader script works, in order to get full credit for your work you have to submit **all** of these files each time that you want to resubmit a solution to either problem.

**Special Note**
Because we're having you write a whole lot of little programs, a `main()` function will not be required for the Short Project 1. But starting with Long Project 1, they will be required!

## 2 `second_largest_of_four_ints.py`

Write a program, in a file named `second_largest_of_four_ints.py` , which calls `input()` four times (not giving the user any prompt). Convert each one to an integer, and then print out the **second largest** of the four values.

You may assume that the input will be four integers (one per line); you don't have to do any error checking.

## 3 call_imported_funcs.py

Write a program, in a file named `call_imported_funcs.py` . Your program must import the file `short1_thing` , which I will provide. You will be calling three functions from inside that file: `foo(), bar(), baz()` .

Your program should first read one line of input from the user. Do not modify it in any way; just read it. Save it for later; you're going to be using it twice.

Second, your program must call the `foo()` function. Pass it exactly one argument - the string you just read. Print out what it returns to you (but also save it for later).

Third, read two more lines of input. Then call `bar()`, and pass it all three of the lines of input you've been given. Print out what this function returns - and save it for later, as well.

Finally, call `baz()`, passing it the return values from both `foo()` and `bar()` - but **do not** call `foo(),bar()` again; instead, pass `baz()` the values you saved. Once again, print out the value that is returned to you.

**NOTE 1:**
The fuctions `foo(),bar(),baz()` may or may not print out something; don't worry about that. You should **only** print out what I've told you to print out, above.

**NOTE 2:**
Do **not** upload your own copy of `short1_thing.py` to GradeScope; the autograder will provide it for you.

**NOTE 3:**
The autograder's version of `short1_thing.py` may be different than the one that I have given you; don't worry! Just call the functions as required, and you will still pass the testcase. After all, isn't the whole point of functions that you **don't need to know** how they work?

## 4 two_lines_of_words.py

Write a program, in a file named `two_lines_of_words.py` , which reads two lines of input from the user. Use `split()` to break each one into words.

Next, print out each of the two arrays of words, along with a little bit of explanatory text (see the example below for details). Then print a blank line. (Remember: To print a blank line, call `print()` with no arguments; it will print nothing, and then follow it up with a newline.)

Next, concatenate the two arrays together, and print out some information about the combined array, followed by another blank line; again, see below for

details.

Next, sort the combined array, and then print it out, followed by a final blank line.

Finally, write a short loop which prints out **pairs** of values - one from the first array, and one from the second array. End the loop when you hit the end of either of the two arrays.

**EXAMPLE:**
Suppose that the user types the following lines of input:

```
asdf      foobar
10 1 11 111 2 -4
```

Your program should produce the following output:

```
The first line was: ['asdf', 'foobar']
The second line was: ['10', '1', '11', '111', '2', '-4']

The combination of both lines had 8 words.
The combined set of words was: ['asdf', 'foobar', '10', '1', '11', '111', '2', '-4']

After sorting, the words were: ['-4', '1', '10', '11', '111', '2', 'asdf', 'foobar']

Pairs:
0: asdf,10
1: foobar,1
```

spec continues on the next page

## 5 `indentation_print.py`

Write a program, in a file named `indentation_print.py` , which reads many lines from the user, in a loop; the easiest way to do this will be an infinite loop:

```
while True:
```

which uses `break` to end the loop when needed.

For each line of input, count how many spaces there were at the beginning of the line, and print that out. (Note that some lines might have no leading spaces, and some might be entirely empty.)

Terminate your loop when the user types `quit` on any input line. Don't terminate the program if the line contains any words other than `quit`; however, you should ignore all whitespace when making this check.

You may assume that the input will always eventually include a `quit` line.

**NOTE:** Do not count the indentation on the `quit` line. Simply break that loop at that point, without counting.

## 6 Turning in Your Solution

You must turn in your code using GradeScope.