CSE 101: Computer Science Principles (Fall 2019)

Lab #5

Assignment Due: Saturday, October 26, 2019, by 11:59 PM

Assignment Objectives

By the end of this assignment you should be able to develop original Python functions to solve simple programming problems involving loops. strings, and lists.

Getting Started

This assignment requires you to write Python code to solve several computational problems. To help you get started, we will give you a basic starter file for each problem. These files will contain *function stubs* and a few tests you can try out to see if your code seems to be correct (**note that the test cases we give you in the starter files are just examples; we will use different test inputs to grade your work!**). You need to complete (fill in the bodies of) these functions for the assignments. **Do not, under any circumstances, change the names of the functions or their parameter lists.** The automated grading system will be looking for functions with those exact names and parameter lists; if you change any function headers, the grading program will reject your solution and mark it as incorrect.

Directions

Solve each of the following problems to the best of your ability. We have provided a code skeleton for each of the programming problems. The automated grading system will execute your solution to each programming problem several times, using different input values each time. Each test that produces the correct/expected output will earn 1 or more points. This assignment contains 3 problems, and is worth a total of 30 points. **Note that not every problem may be worth the same number of points!**

- ⚠ Each starter file has a comment block at the top with various important information. Be sure to add your information (name, ID number, and NetID) to the first three comment lines, so that we can easily identify your work. **If you leave this information out, you may not receive credit for your work!**
- ⚠ Submit your work as a set of individual files (one per problem). **DO NOT** zip or otherwise compress your files, and **DO NOT** place all of your work in a single file. If you do so, we may not be able to grade your work and you will receive a failing grade for this assignment!
- ⚠ Every function **MUST** use the names and parameter lists indicated in the starter code file. Submissions that have the wrong function names (or whose functions contain the wrong number of parameters) can't be graded by our automated grading system, and may receive a grading penalty (or may not be graded at all).
- ⚠ Every function must explicitly ***return*** its final answer; the grading program will ignore anything that your code prints out. Along those lines, do **NOT** use `input()` anywhere within your functions (or anywhere before the `if __name__ == "__main__":` statement); your functions should get all of their input from their parameters. Programs that crash will likely receive a failing grade, so test your code thoroughly **with Python 3.7.4 or later** before submitting it.
- ⚠ Blackboard will provide information on how to submit this assignment. You **MUST** submit your completed work as directed by the indicated due date and time. We will not accept (or grade) any work that is submitted after the due date and time, or that is submitted before you have signed the course's Academic Honesty agreement.
- ⚠ **ALL** of the material that you submit (for each problem) **MUST** be your own work! You may not receive assistance from or share any materials with anyone else, except for the instructor and the (current) course TAs.

Part I: Password Validation (10 points)

(Place your answer to this problem in the "validate_password.py" file)

Many computer systems compare a potential password to a series of rules to determine whether or not it is valid. They accept a string representing a password, and then report whether or not that password is valid.

Complete the `validate_password()` function, which takes in one string argument (the password to check). This function should check if the string follows the rules listed below in order to determine whether it is a valid password or not. If the password is valid (i.e., it satisfies **all** of the rules below), the function should return the string `"valid"`; otherwise, it should return the string `"invalid"`.

PASSWORD RULES

- The password must contain at least 8 characters
- The password must contain at least one binary digit (either a 0 or a 1) (other digits in the range 2–9 are permitted as long as at least one binary digit is also present)
- The password must contain at least one lowercase letter (a–z)
- The password must contain at least one uppercase letter (A–Z)
- The password must contain at least one special character from the set !, @, #, $, and %

Examples:

Function Call	Return Value	Notes
`validate_password("SBCs123")`	`invalid`	Too short, does not contain a special character
`validate_password("SBcs9876")`	`invalid`	Does not contain a binary digit or a special character
`validate_password("CSE101!isCool")`	`valid`	
`validate_password("cse101!iscool")`	`invalid`	Does not contain any uppercase letters
`validate_password("CSE101@ISCOOL")`	`invalid`	Does not contain any lowercase letters
`validate_password("CSE337%usesPerl")`	`invalid`	Does not contain a 0 or a 1

Part II: Just the Necessities (10 points)

(Place your answer to this problem in the "necessities.py" file)

Every week, a train carrying various items visits the village of Tinyville to drop off any items that the town may need (if it has them). When the train leaves, it should contain any items that the town did not need that week.

Complete the `necessities()` function, which takes two arguments: a list of strings that represents all of the items carried by the train, and a list representing items that the town needs (both lists may contain duplicate items). The function returns an integer value representing the total number of items that were dropped off at the town.

NOTE: Every train needs a conductor. If the train does not contain the string "conductor" as one of its elements, your function should return the string "Warning: no conductor" (be sure to match the spelling and capitalization, and place **EXACTLY ONE** space between each word) instead of an integer value.

Hint: You may find it useful to use a combination of the `index()` and `del` commands. Alternately, look up Python's `remove()` command.

Examples:

Function Call	Return Value
`necessities(["conductor", "flowers", "grain"], ["pots", "flowers", "wood"])`	`1`
`necessities(["conductor", "clay", "ceramic", "silver", "copper"], ["pots", "wool"])`	`0`
`necessities(["flowers", "diamonds"], ["pots", "flowers", "wood", "iron"])`	Warning: no conductor
`necessities(["flowers", "conductor", "diamonds", "diamonds", "flowers"], ["pots", "diamonds", "wood", "iron", "diamonds"])`	`2`

Part III: Error-Checking Base Conversion (10 points)

(Place your answer to this problem in the "conversion.py" file)

Complete the `conversion()` function, which takes two arguments: a string containing a number in either binary (base 2) or octal (base 8) format, and an integer representing that number's base (for example, if the first argument is a binary number, the second argument should be 2).

- if the specified base is not 2 or 8, the function should return the all-lowercase string "illegal base".
- If the binary or octal number contains a digit that is invalid for that base (for example, a "binary" number that contains a 3, the function should return the all-lowercase string "illegal digit". You may assume that the number to be converted will only contain the digits 0–9, not letters or special characters.
- Otherwise, the function should convert the input to decimal (base 10) and return that value (as an integer, not a string).

No other output should be returned (for example, additional explanatory text).

Examples:

Function Call	Return Value	Notes/Explanation
`conversion("101", 2)`	5	
`conversion("112", 2)`	illegal digit	2 is not a valid binary digit
`conversion("073", 8)`	59	
`conversion("821", 8)`	illegal digit	8 is not a valid octal digit
`conversion("456", 4)`	illegal base	This program only handles base 2 and base 8