# CMSC 201 Spring 2018
## Homework 5 –Lists and Strings

**Assignment:** Homework 5 – Lists and Strings
**Due Date:** Friday, October 12th, 2018 by 8:59:59 PM
**Value:** 40 points

**Collaboration:** For Homework 5, collaboration is allowed. Make sure to consult the syllabus about the details of what is and is not allowed when collaborating. You may not work with any students who are not taking CMSC 201 this semester.

If you work with someone, remember to note their name, email address, and what you collaborated on by filling out the Collaboration Log.

You can find the Collaboration Log at https://tinyurl.com/collab201-fa18.

Remember that all collaborators need to fill out the log each time; even if the help was only "one way" help.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
# File:     FILENAME.py
# Author:   YOUR NAME
# Date:     THE DATE
# Section:  YOUR DISCUSSION SECTION NUMBER
# E-mail:   YOUR_EMAIL@umbc.edu
# Description:
#    DESCRIPTION OF WHAT THE PROGRAM DOES
```

## Instructions

For each of the questions below, you are given a problem that you must solve or a task you must complete.

This assignment will focus on using functions to make code that is more modular. Some of these functions may be useful in later assignments!

**At the end, your Homework 5 files must run without any errors.**

**NOTE: Your filenames for this homework must match the given ones exactly.**
And remember, filenames are case sensitive!

## Additional Instructions – Creating the hw5 Directory

During the semester, you'll want to keep your different Python programs organized, organizing them in appropriately named folders (also known as directories).

Just as you did for previous homeworks, you should create a directory to store your Homework 5 files. We recommend calling it **`hw5`**, and creating it inside the **`Homeworks`** directory inside the **`201`** directory.

If you need help on how to do this, refer back to the detailed instructions in Homework 1. (You don't need to make a separate folder for each file. You should store all of the Homework 5 files in the same **`hw5`** folder.)

## Coding Standards

Prior to this assignment, you should re-read the Coding Standards, available on Blackboard under "Assignments" and linked on the course website at the top of the "Assignments" page.

For now, you should pay special attention to the sections about:

- Constants
  - For Homework 5, you must use constants instead of magic numbers!!! Magic strings are also forbidden!!!!!!
- Make sure to **read the last page of the Coding Standards document**, which prohibits the use of certain tools and Python keywords
  - Also, this section states that **you may not use built-in functions** or concepts we haven't covered in class!
    If you use a built-in function to solve a problem, you will earn **zero points** for that entire problem.

## Additional Specifications

For this assignment, **you must use** `main()` as seen in class.

---

For this assignment, you should pay attention to each problem's instructions on using "input validation." For example, the user may enter a negative value, but your program may require a positive value. **Make sure to follow each part's instructions about input validation.**

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

---

You must create the functions specified in each problem, and they must be named ***exactly*** as shown.

## Questions

Each question is worth the indicated number of points. Following the coding standards is worth 4 points. If you do not have complete file headers and correctly named files, you will lose points.

**hw5_part1.py** **(Worth 8 points)**

For this part of the homework you will create a program that displays a number line to the user based on a sequence of numbers that they enter.

The user will enter a sequence of numbers (-1 to stop). **An entered number is only valid if it is larger than every accepted number beforehand.** See the sample output for an example. You should use a list to store accepted numbers! This will give a single place where all the numbers entered exist.

Once you have this list of numbers, you should then output a number line that starts at 0 and stops at the last entered number. If the user entered a number in this range, you should print that number on the number line. If they did not enter a number in that range, you should output "-" in its place.

There is sample output on the next page.

*(HINT: while loops, just like if statements, can be nested!)*

***You should not use ANY built in Python functions that we have not covered in class. This means no min(), max(), count(), etc.***

***Failure to follow this rule will result in a score of 0.***

Here is some sample output, with the user input in **blue**.
(Your output should match exactly!)

```
Michaels-MacBook-Pro-6:hw5 mneary1$ python3 hw5_part1.py
Please enter a number (-1 to stop): 0
Please enter a number (-1 to stop): 2
Please enter a number (-1 to stop): 4
Please enter a number (-1 to stop): 6
Please enter a number (-1 to stop): -1
0 - 2 - 4 - 6

Michaels-MacBook-Pro-6:hw5 mneary1$ python3 hw5_part1.py
Please enter a number (-1 to stop): 1
Please enter a number (-1 to stop): 2
Please enter a number (-1 to stop): 4
Please enter a number (-1 to stop): 5
Please enter a number (-1 to stop): 7
Please enter a number (-1 to stop): 8
Please enter a number (-1 to stop): -1
- 1 2 - 4 5 - 7 8

Michaels-MacBook-Pro-6:hw5 mneary1$ python3 hw5_part1.py
Please enter a number (-1 to stop): 5
Please enter a number (-1 to stop): 10
Please enter a number (-1 to stop): 15
Please enter a number (-1 to stop): 20
Please enter a number (-1 to stop): -1
- - - - - 5 - - - - 10 - - - - 15 - - - - 20

Michaels-MacBook-Pro-6:hw5 mneary1$ python3 hw5_part1.py
Please enter a number (-1 to stop): 4
Please enter a number (-1 to stop): 2
Please enter a bigger number
Please enter a number (-1 to stop): 7
Please enter a number (-1 to stop): 3
Please enter a bigger number
Please enter a number (-1 to stop): 6
Please enter a bigger number
Please enter a number (-1 to stop): 8
Please enter a number (-1 to stop): -1
- - - - 4 - - 7 8
```

**hw5_part2.py** **(Worth 6 points)**

This part of the homework you will be creating new words from a list of words that the user enters. The user will enter exactly 5 words to your program. These words are going to be exactly 5 characters long. Once you have gotten all 5 words from the user, you will use specific letters from each words to show a new one.

You must validate the words entered by the user. You must ensure that each word is exactly 5 characters long. If any word is not exactly 5 characters long, you must reprompt the user until they enter a correct word.

The process of generating the new word for the user is simple. The new word is a combination of a single letter from each word entered by the user. The *first letter of the new word* is the first letter of the first word entered. The *second letter of the new word* is the second letter of the second word entered by the user. You should see the pattern here… the new word is found on the diagonal if I were to make a square out of the entered words!

*HINT: Since you can have strings inside of lists in Python, that means you can have "nested" indexing. If you have a list of strings, and you want to get the first character of the first string in that list you would write something like:*

```
listOfStrings = ["kiwi", "banana", "pear"]
print(listOfStrings[0][0]) # would print the letter "k"
```

You will find sample output on the next page.

Here is some sample output, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
linux1[4]% python3 hw5_part2.py
Enter a 5-letter word: fizzy
Enter a 5-letter word: jumpy
Enter a 5-letter word: juicy
Enter a 5-letter word: mixup
Enter a 5-letter word: fuzed

Your new word is: fuiud

linux1[5]% python3 hw5_part2.py
Enter a 5-letter word: deadly
Must be 5 letters!
Enter a 5-letter word: banana
Must be 5 letters!
Enter a 5-letter word: admit
Enter a 5-letter word: bambi
Enter a 5-letter word: bakes
Enter a 5-letter word: darts
Enter a 5-letter word: debug

Your new word is:aaktg

linux1[6]% python3 hw5_part2.py
Enter a 5-letter word: fluid
Enter a 5-letter word: baked
Enter a 5-letter word: apples
Must be 5 letters!
Enter a 5-letter word: pears
Enter a 5-letter word: axe
Must be 5 letters!
Enter a 5-letter word: lambs
Enter a 5-letter word: fizzy

Your new word is: faaby
```

**hw5_part3.py** **(Worth 10 points)**

Write a program that asks the user to enter a password, and then checks it for a few different requirements before approving it as secure and repeating the final password to the user.

The program must re-prompt the user until they provide a password that satisfies all of the conditions. It must also tell the user each of the conditions they failed, and how to fix it.
If there is more than one thing wrong (*e.g.*, no lowercase, and longer than 20 characters), the program must print out all of the things that are wrong, and how to fix them.

The program follows these rules for passwords:

1. The password must contain at least one lowercase letter.
2. The password must contain at least one uppercase letter.
3. The password must be between 6 and 20 characters, inclusive.
    a. If the password is between 6 and 13 characters, inclusive, it must contain a "7" somewhere in the password.
    b. If the password is between 14 and 20 characters, inclusive, it must contain a "2" somewhere in the password.
4. The password cannot contain the characters "l" and "I" (lowercase "L" and uppercase "i") at the same time. (It can contain either, just not both at the same time. It may also contain neither.)

For this part of the homework, you ***must*** have an in-line comment at the top of **each** of your program's individual `if`, `elif`, and `else` statements, explaining what is being checked by that conditional.

*(HINT: Think carefully about what your conditionals should look like. If necessary, draw a truth table to help figure out what different inputs will do. Using a Boolean flag will also likely make this easier.)*

(See the next page for sample output.)

```
linux2[3]% python3 hw5_part3.py
Please enter a password: dogs
Password must have an uppercase character
Password must be at least 6 characters
Please enter a password: DOGS
Password must have a lowercase character
Password must be at least 6 characters
Please enter a password: Dogs
Password must be at least 6 characters
Please enter a password: Doggos
Shorter passwords must contain a 7
Please enter a password: 7Doggos
Thank you for picking the secure password 7Doggos

linux3[4]% python3 hw5_part3.py
Please enter a password: thisMustBeSecureItsLongAlso27
Password must be no longer than 20 characters
Password cannot contain a I and a l at the same time
Please enter a password: abcdefghijklmnopqrst
Password must have an uppercase character
Longer passwords musts contain a 2
Please enter a password: 2and7EQUALSnine
Thank you for picking the secure password 2and7EQUALSnine

linux2[5]% python3 hw5_part3.py
Please enter a password: I_and_l
Shorter passwords must contain a 7
Password cannot contain an I and an l at the same time
Please enter a password: I_and_7
Thank you for picking the secure password I_and_7

linux2[6]% python3 hw5_part3.py
Please enter a password: greatPassword7!
Longer passwords must contain a 2
Please enter a password: greatPassword2!
Thank you for picking the secure password greatPassword2!
```

**hw5_part4.py** **(Worth 8 points)**

Create a program that plays with the capitalization of letters in a given string. The program should have four different ways to manipulate the capitalization of a string, and those options should be displayed as a menu. First you should get the menu choice from the user, then you should accept a string from the user to apply the operation to. Display the string after applying whatever operation that was chosen.

The four different options are: making the string all uppercase, making the string all lowercase, making every other character uppercase, and inverting the capitalization (upper becomes lower, and lower becomes upper).

Make sure to validate the menu option, you must ensure the user enters a correct choice (between 1-4 inclusive). You can assume that no special characters will be entered – only English letters will be used in the strings (no spaces, etc).

See sample output on the next page.

Here is some sample output, with the user input in **blue**.
**(Your output must match exactly!)**

```
linux2[3]% python3 hw5_part4.py
1 - All uppercase.
2 - All lowercase.
3 - Every other letter.
4 - Opposite.
Please enter a menu option: 1
Please enter a string: example
EXAMPLE

linux2[4]% python3 hw5_part4.py
1 - All uppercase.
2 - All lowercase.
3 - Every other letter.
4 - Opposite.
Please enter a menu option: 2
Please enter a string: NOTYELLING
notyelling

linux2[5]% python3 hw5_part4.py
1 - All uppercase.
2 - All lowercase.
3 - Every other letter.
4 - Opposite.
Please enter a menu option: 3
Please enter a string: kindergarten
KiNdErGaRtEn

linux2[6]% python3 hw5_part4.py
1 - All uppercase.
2 - All lowercase.
3 - Every other letter.
4 - Opposite.
Please enter a menu option: 5
Invalid Choice!
Please enter a valid menu option: 0
Invalid Choice!
Please enter a valid menu option: 4
Please enter a string: InVeRtMe
iNvErTmE
```

**hw5_part5.py** **(Worth 4 points)**

**[REDACTED]**

**No validation of input is required. Turn in the code that you used to solve this puzzle!**

**[REDACTED]**

```
linux3[5]% python3 hw5_part5.py
Enter the secret message:
dbvuoezSBgcohrXostQkdMeiVeRYfjtaJvchSdixeXnLimpgZSerMCcsHswWsbcEMa
xFusgLUdUewBxX
Enter the key to this message: 5

decodethemessage

linux3[6]% python3 hw5_part5.py
Enter the secret message: iItvidseeNabsEyv
Enter the key to this message: 2

itiseasy

linux3[7]% python3 hw5_part5.py
Enter the secret message:
ikPHfwXuyJSeoMuJuUAdcoQOavXKnRUTsAxieRQDebiGtCxzhcbDeDdqpPSQaorWtO
VrtEhmeRDUrEQenkca
Enter the key to this message: 4

ifyoucanseethepattern

linux3[8]% python3 hw5_part5.py
Enter the secret message:
caeoDbuhmnretgRcqBaRfrlNeaLfQcuXZlQhlJDyhf
Enter the key to this message: 3

countcarefully
```

**yUDdVsLWoFKHkxgLuFrNKUTnhOyonlsCazkrMuswvCsLBjcWeMZkYkSYcwkGDUF
ertSNXFetacRgiXKOcKePRrHKkZUlKNzsebeoJTBHdzbRYozztcnlBSGSheKXsGuveV
PsGtmFcgKQxlDDonKUGgdqdKzLUtgFeojzeOik
8**

## Submitting

Once your **hw5_part1.py**, **hw5_part2.py**, **hw5_part3.py**, **hw5_part4.py**, and **hw5_part5.py** files are complete, it is time to turn them in with the **submit** command. (You may also turn in individual files as you complete them. To do so, only **submit** those files that are complete.)

You must be logged into your account on GL, and you must be in the same directory as your Homework 5 Python files. To double-check you are in the directory with the correct files, you can type **ls**.

```
linux1[3]% ls
hw5_part1.py  hw5_part3.py  hw5_part2.py  hw5_part4.py
hw5_part5.py
linux1[4]%
```

To submit your Homework 5 Python files, we use the **submit** command, where the class is **cs201**, and the assignment is **HW5**. Type in (all on one line) **submit cs201 HW5 hw5_part1.py hw5_part2.py hw5_part3.py hw5_part4.py hw5_part5.py** and press enter.

```
linux1[4]% submit cs201 HW5 hw5_part1.py hw5_part2.py
hw5_part3.py hw5_part4.py
Submitting hw5_part1.py...OK
Submitting hw5_part2.py...OK
Submitting hw5_part3.py...OK
Submitting hw5_part4.py...OK
Submitting hw5_part5.py...OK
linux1[5]%
```

If you don't get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can check that your homework was submitted by following the directions in Homework 0. Double-check that you submitted your homework correctly, since **an empty file will result in a grade of zero for this assignment.**