## CMSC 201 Section 40

# Homework 3 - Lists and Loops

**Due Date:** Monday, March 7th, 2021 by 11:59:59 PM
**Value:** 40 points

**This assignment falls under the standard cmsc201 academic integrity policy. This means you should not discuss/show/copy/distribute your solutions, your code or your main ideas for the solutions to any other student. Also, you should not post these problems on any forum, internet solutions website, etc.**

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
"""
```

# Creating Your HW3 Directory

```
linux3[1]% cd cmsc201/Homeworks
linux3[2]% mkdir hw3
linux3[3]% cd hw3
linux3[4]%
```

# Submission Details

Submit the files under the following titles:
(These are case sensitive as usual. )
submit cmsc201 HW3 {files go here}

| | |
|---|---|
| Problem 1 - Log of Two | log_of_two.py |
| Problem 2 - Wordle Checker | wordle_checker.py |
| Problem 3 - Draw an X | draw_an_x.py |
| Problem 4 - List Merge | list_merge.py |
| Problem 5 - Which Month II | which_month_2.py |

For example you would do:

```
linux1[4]% submit cmsc201 log_of_two.py wordle_checker.py
draw_an_x.py list_merge.py which_month_two.py
Submitting log_of_two.py...OK
Submitting wordle_checker.py...OK
Submitting draw_an_x.py...OK
Submitting list_merge.py...OK
Submitting which_month_2.py...OK
linux1[5]%
```

From here, you can use **`emacs`** to start creating and writing your different Homework 3 Python programs.

You don't need to and should not make a separate folder for each file. You should store all of the Homework 3 files in the same **`hw3`** folder.

# Coding Standards

Coding standards for CMSC 201 can be found here.

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

- Naming Conventions
- Use of Whitespace
- Comments (specifically, File Header Comments)
- Variable names.

Make sure to include **`if __name__ == '__main__':`** at the beginning of each file and indent your code inside it.

# Input Validation

For this assignment, you do **not** need to worry about **SOME** “input validation.”

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

Unlike in HW1 you didn't have to worry about any input validation since you didn't have if statements, but now you do, so you can worry a little about it. You can try to prevent invalid input, but only when that invalid input is of the correct type.

For example, if your program asks the user to enter a whole number, it is acceptable if your program crashes if they enter something else like “dog” or “twenty” or “88.2” instead.

But it's a good idea to try to catch a zero division error for instance, or entering negative numbers when only positive numbers are allowed.

Here is what that error might look like:

```
Please enter a number: twenty
Traceback (most recent call last):
  File "test_file.py", line 10, in <module>
    num = int(input("Please enter a number: "))
ValueError: invalid literal for int() with base 10: 'twenty'
```

# Allowed Built-ins/Methods/etc

- Declaring and assigning variables, ints, floats, bools, strings.
- Casting int(x), str(x), float(x), (technically bool(x))
- Using +, -, *, /, //, %, **; +=, -=, *=, /=, //=, %=, **= where appropriate
- Print, with string formatting, with end= or sep=:
    - '{}'.format(var), '%d' % some_int, f-strings
    - Really the point is that we don't care how you format strings in Python
    - Ord, chr, but you won't need them this time.
- Input, again with string formatting in the prompt, casting the returned value.
- Using the functions provided to you in the starter code.
- Comparisons ==, <=, >=, >, <, !=, in
- Logical and, or, not
- if/elif/else, nested if statements
- Using import with libraries and specific functions **as allowed** by the project/homework.
- String Operations:
    - upper(), lower()
    - concatenation +, +=
- For loops, both *for i* and *for each* type.
- Lists, list(), indexing, i.e. my_list[i] or my_list[3]
    - 2d-lists if you want them/need them my_2d[i][j]
    - Append, remove
- split(), lower(), upper() for strings
- len() for strings and lists

# Forbidden Built-ins/Methods/etc

This is not a complete listing, but it includes:

While loops

- sentinel values, boolean flags to terminate while loops

**list slicing**

If you have read this section, then you know the secret word is: adventurous.

String operations, strip(), join(), isupper(), islower()

- **string slicing**

**Dictionaries**

- creation using dict(), or {}, copying using dict(other_dict)
- .get(value, not_found_value) method
- accessing, inserting elements, removing elements.

break, continue

methods outside those permitted within allowed types

- for instance str.endswith
- list.index, list.count, etc.

Keywords you definitely don't need: await, as, assert, async, class, except, finally, global, lambda, nonlocal, raise, try, yield

The *is* keyword is forbidden, not because it's necessarily bad, but because it doesn't behave as you might expect (it's not the same as ==).

built in functions: any, all, breakpoint, callable, classmethod, compile, exec, delattr, divmod, enumerate, filter, map, max, min, isinstance, issubclass, iter, locals, oct, next, memoryview, property, repr, reversed, round, set, setattr, sorted, staticmethod, sum, super, type, vars, zip

exit() or quit()

If something is not on the allowed list, not on this list, then it is probably forbidden.

The forbidden list can always be overridden by a particular problem, so if a problem allows something on this list, then it is allowed for that problem.

# Problem 1 - Log of Two

The natural log of 2 is approximately 0.693

We are going to calculate better approximations using python.

We will use the summation:

$$\sum_{i=1}^{\infty} \frac{1}{i2^i}$$

Naturally, as computer scientists, we cannot expect a computer to take a sum to infinity, so we'll have to cut off at a specific n, which you will input.

$$\sum_{i=1}^{n} \frac{1}{i2^i}$$

For instance, if we set n = 3, then we'll add up:

$1/(1 * 2^1) + 1/(2 * 2^2) + 1/(3 * 2^3) = ½ + ⅛ + 1/24 = 0.666$

For instance, if we set n = 5, then we'll add up:

$1/(1 * 2^1) + 1/(2 * 2^2) + 1/(3 * 2^3) + 1/(4 * 2^4) + 1/(5 * 2^5) = ½ + ⅛ + 1/24 + 1/64 + 1/160 = 0.688541$ (approximately).

As we increase n, we get closer estimates to the correct answer.

```
linux[0]$ python3 log_of_two.py
Enter the number of terms to sum: 5
After 5 terms the ln(2) = 0.6885416666666666

linux[1]$ python3 log_of_two.py
Enter the number of terms to sum: 100
After 100 terms the ln(2) = 0.6931471805599451

linux[2]$ python3 log_of_two.py
Enter the number of terms to sum: 2
After 2 terms the ln(2) = 0.625

linux[3]$ python3 log_of_two.py
Enter the number of terms to sum: 17
After 17 terms the ln(2) = 0.6931467770529169

linux[4]$ python3 log_of_two.py
Enter the number of terms to sum: 10000
After 10000 terms the ln(2) = 0.6931471805599451
```

# Problem 2 - Wordle Checker

Create a program in a file called wordle_checker.py.

[No spoilers to the recent wordles occur in this problem.]

You should enter a "solution word" and then check a "test word" which the user will also input. The words should be the same length, check to ensure that this is the case.

The way that wordle works is that if for instance the solution word is "chafe" and we enter "ready" then the result will look like:


The reason is that the a is in the correct position, and there is an 'e' in the word but it's not in the right position.

For instance, if the word is:


and our guess was:


Then we see that there are four yellow positions because there are four letters that match but are not in the correct position.

We will implement this with one primary difference however, to simplify the problem, any letter which is in the word but doesn't have the right position will be marked yellow, even if there aren't enough of those letters to match.

For instance if the solution word was party and we guessed rrrrr, which in the regular game is not allowed since it's not a word, but in our checker will be allowed, the result should be:


To output the results, output '_' for no match or gray, 'y' for yellow and 'g' for green. So the result should be yygyy.

You can either use a list to store it and just output a list as the result or output the string result at the end.

A result like:


should be printed out as:

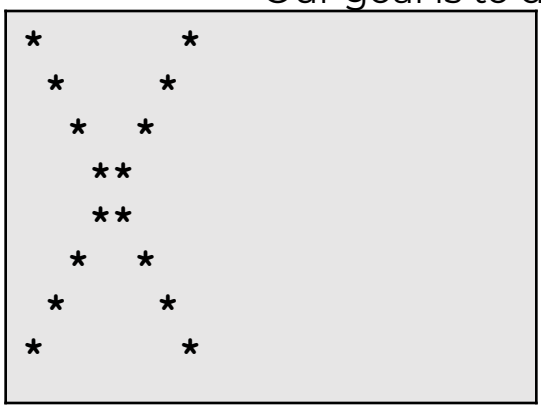
yyy_y.

```
linux[0]$ python3 wordle_checker.py
Enter the solution word: party
Enter the guess word: align
y _ _ _ _

linux[1]$ python3 wordle_checker.py
Enter the solution word: silly
Enter the guess word: lapel
y _ _ _ y

linux[2]$ python3 wordle_checker.py
Enter the solution word: stain
Enter the guess word: steam
g g _ y _

linux[3]$ python3 wordle_checker.py
Enter the solution word: games
Enter the guess word: sages
y g y g g
```

# Problem 3 - Draw an X

Write this program in a file called draw_an_x.py

Our goal is to draw shapes like this one:




In your program you should ask for a size and then you should draw a "square grid" of stars and spaces, where they end up drawing the stars in an x pattern.

Hint 1: You will have to print out some blank spaces as well as stars.

Hint 2: In order to print a star or space without a newline, use the parameter end="" or end='' at the end of your print statement like this:

```
    print('*', end='')
```

Hint 3: Now that you've prevented the endlines from printing, if you want just an endline you can use:

```
    print()
```

Sample output:

User input is generally colored **blue** to help distinguish it from the rest of the text.

```
linux4[120]% python3 draw_an_x.py
What is the size of the X that we want to draw? 12
*           *
 *         *
  *       *
   *     *
    *   *
     **
     **
    *   *
   *     *
  *       *
 *         *
*           *

linux4[121]% python3 triangle.py
What is the height/width of the triangle? 9
*       *
 *     *
  *   *
   * *
    *
   *   *
  *     *
 *       *
*         *
linux4[122]% python3 triangle.py
What is the height/width of the triangle? 3
* *
 *
* *
```

# Problem 4 - List Merge

To "merge" two lists, we want to interleave them, so for instance, if we have the lists [1, 2] and [3, 4] the merge of them will be:

[1, 3, 2, 4]

Another example:
['a', 's', 'd', 'f'] and ['a', 'b', 'c', 'd'] will result in:

['a', 'a', 's', 'b', 'd', 'c', 'f' , 'd']

Write a program that takes in two lists with the same number of elements and then outputs them in the following way.

I want you to output the list itself, don't worry about pretty formatting, because I want to see that the internal lists have in fact been merged into a new list.

DO NOT USE: zip (it won't produce the correct result anyway).

Sample Output

```
linux[0]$ python3 list_merge.py
How  many elements do you want in each list? 3
What do you want to put in the first list? a
What do you want to put in the first list? b
What do you want to put in the first list? c
What do you want to put in the second list? 1
What do you want to put in the second list? 2
What do you want to put in the second list? 3
The first list is:  ['a', 'b', 'c']
The second list is:  ['1', '2', '3']
The merged list is:  ['a', '1', 'b', '2', 'c', '3']

linux[0]$ python3 list_merge.py
How  many elements do you want in each list? 3
What do you want to put in the first list? How
What do you want to put in the first list? you
What do you want to put in the first list? today
What do you want to put in the second list? do
What do you want to put in the second list? feel
What do you want to put in the second list? ?
The first list is:  ['How', 'you', 'today']
The second list is:  ['do', 'feel', '?']
The merged list is:  ['How', 'do', 'you', 'feel', 'today',
'?']
```

# Problem 5 - Which Month II

Name your file which_month_2.py

This time, you'll be redoing the problem from the previous homework, so the input and output will be the same, with the exception that you are **required to use a list** rather than 12 if/elif/else statements.

In order to prevent this, we'll establish it as a rule that you **cannot use more than 3** if/elif/else statements for this problem.

Here are the months along with their corresponding numbers again:

| Month | Number |
|---|---|
| January | 1 |
| February | 2 |
| March | 3 |
| April | 4 |
| May | 5 |
| June | 6 |
| July | 7 |
| August | 8 |
| September | 9 |
| October | 10 |
| November | 11 |
| December | 12 |

First ask what month it is "now" (m) and then ask how many months into the future you want to go (n). These should both be integers. Then display what month it is in the future n months after m.

Display the answer as the actual name of the month. The number of months after the start can be more than 12. [Hint: use mod]

**Check to see if the first input is between 1 and 12 before continuing.**

Sample Output:

```
linux3[14]% python3 which_month_2.py
What month are we starting in (enter as an int)? 11
How many months in the future should we go? 24
The month will be November

linux3[15]% python3 which_month_2.py
What month are we starting in (enter as an int)? 5
How many months in the future should we go? 1
The month will be June

linux3[16]% python3 which_month_2.py
What month are we starting in (enter as an int)? 8
How many months in the future should we go? 71
The month will be July

linux3[17]% python3 which_month_2.py
What month are we starting in (enter as an int)? 3
How many months in the future should we go? 9
The month will be December

linux3[18]% python3 which_month_2.py
What month are we starting in (enter as an int)? 3
How many months in the future should we go? 10
The month will be January

linux3[19]% python3 which_month_2.py
What month are we starting in (enter as an int)? 17
That is not a month between 1 and 12.
```