Homework 3 - Lists and Loops
CMSC 201 Section 60

Due Date: Monday, October 2nd, 2023 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 - Basel Problem	basel.py
Problem 2 - Diphthong Detector	detector.py
Problem 3 - Draw a Square	draw_a_square.py
Problem 4 - List Merge	list_merge.py
Problem 5 - Day of the Week II	day_2.py

For example you would do:

```
linux1[4]% submit cmsc201 HW3 slice_of_pi.py wordle_checker.py
draw_a_square.py list_merge.py day_2.py
Submitting basel.py...OK
Submitting detector.py...OK
Submitting draw_a_square.py...OK
Submitting list_merge.py...OK
Submitting day_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
    - .join()
- 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 - Basel Problem

The following sum is a well known problem with the solution dating back to Euler.

$$\sum_{n=1}^{\infty} \frac{1}{n^2}$$

However it's an infinite sum so we cannot actually use a computer to sum it without stopping at some point. Ask the user how far they want to go, and calculate the sum up to that point. For instance, if they ask for 5 terms we want to add up 1/1 + 1/4 + 1/9 + 1/16 + 1/25.

```
linux[0]$ python3 basel.py
What is the number of terms you want to sum? 100
The approximation for 100 terms is 1.6349839001848923

linux[1]$ python3 basel.py
What is the number of terms you want to sum? 2500
The approximation for 2500 terms is 1.6445341468375643

linux[2]$ python3 basel.py
What is the number of terms you want to sum? 10000
The approximation for 10000 terms is 1.6448340718480652

linux[3]$ python3 basel.py
What is the number of terms you want to sum? 100000
The approximation for 100000 terms is 1.6449240668982423

linux[4]$ python3 basel.py
What is the number of terms you want to sum? 5
The approximation for 5 terms is 1.4636111111111112
```

Problem 2 - Diphthong Detector

Create a program in a file called detector.py. (Just call it detector.py to avoid having to spell diphthong.)

Diphthongs are also known as gliding vowels, i.e. a combination of two adjacent vowel sounds within the same syllable.

For instance loud is the combination of the sounds aʊ.

Similarly, lair is the combination of ɛə.

We're going to approximate this by assuming that if two vowels are next to each other then they produce a diphthong. So even though it may not always be a perfect match our goal is to detect pairs of vowels.

We will ignore any u following a q. So for instance Quiet will only have one because the Qu will be ignored and the ie will count.

For instance if we were going to count the number of diphthongs in beer, bear, pair, day [y is a vowel for our purposes] then you should say 1.

For a word like c[ou]rth[ou]se you should say 2. If we try a word like queueing then it can get complicated because there are multiple possible ways to pair. There are 3 pairs in this word, eu, ue, ei, technically there is also the initial ue but that u follows a q.

Hint: Don't worry about the qu rule at first, just count up the number of pairs of vowels first. If a vowel is at position i, then where does the next vowel have to be?

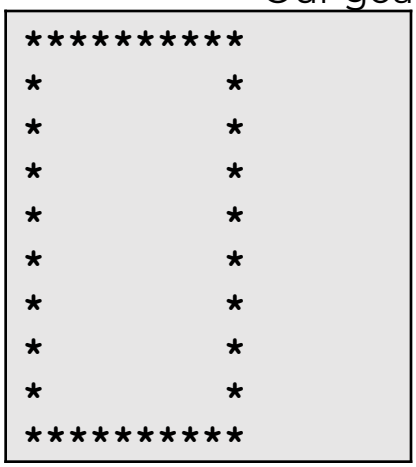
Sample Output

```
linux[0]$ python detector.py
Tell me the word you wish to check for diphthongs:
question
There are 1 diphthongs in the string

linux[1]$ python detector.py
Tell me the word you wish to check for diphthongs: placate
There are 0 diphthongs in the string

linux[2]$ python detector.py
Tell me the word you wish to check for diphthongs:
courthouse
There are 2 diphthongs in the string

linux[3]$ python detector.py
Tell me the word you wish to check for diphthongs:
enqueueing
There are 3 diphthongs in the string

linux[4]$ python detector.py
Tell me the word you wish to check for diphthongs: feet
There are 1 diphthongs in the string
```

Problem 3 - Draw a Square

Write this program in a file called draw_a_square.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.

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_a_square.py
What is the size of the square that we want to draw? 10
**********
*        *
*        *
*        *
*        *
*        *
*        *
*        *
*        *
**********



linux4[121]% python3 draw_a_square.py
What is the size of the square that we want to draw? 7
*******
*      *
*     *
*      *
*      *
*      *
*******
linux4[122]% python3 draw_a_square.py
What is the size of the square that we want to draw? 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 asks the user how many elements each list should have [ask once since it will be the same number.] Then the program should take in two lists with the same number of elements. Finally, output them in the following way as shown in the sample output.

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.

Call your file list_merge.py

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 - Day of the Week II

Name your file day_2.py

This time you must determine the day of the week using at least one list and modulus division.

We're going to ask for the numerical date in September 2023, for instance September 16th. We will tell the user that September 16th is a Saturday.

The first day of September is a Friday, and there are 30 days in the month total.

You should check that the user enters a number between 1 and 30 [inclusively] so that they don't go out of bounds of the month.

You should then tell the user what the day of the week is.

As an additional challenge I want you to also get the suffix right, so for instance, if the day is 1, 21 then it should say "1st" and "21st". If it is 2nd or 22nd, you should use the 'nd' suffix, and if the day is 3 or 23 you should add 'rd' as the suffix. Otherwise the suffix should be 'th'.

You should be using modulus division for this problem.

You can reuse some of your code for determining the suffix, but you must change the code that determines the day of the week.

Sample Output:

```
linux3[14]% python3 day_of_the_week.py
What day of September 2023 is it? 13
September 13th is a Wednesday

linux3[15]% python3 day_of_the_week.py
What day of September 2023 is it? 21
September 21st is a Thursday

linux3[16]% python3 day_of_the_week.py
What day of September 2023 is it? 29
September 29th is a Friday

linux3[17]% python3 day_of_the_week.py
What day of September 2023 is it? 2
September 2nd is a Saturday

linux3[18]% python3 day_of_the_week.py
What day of September 2023 is it? 1
September 1st is a Friday

linux3[19]% python3 day_of_the_week.py
What day of September 2023 is it? 77
That day 77 is out of range, you must enter a number
between 1 and 30

linux3[20]% python3 day_of_the_week.py
What day of September 2023 is it? -3
That day -3 is out of range, you must enter a number
between 1 and 30
```