ASSIGNMENT 3

COMP202, Winter 2021

Due: Tuesday, April 13th, 11:59pm

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	35 points
Question 2:	65 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure.

To get full marks, you must:

- Follow all directions below.
  - In particular, make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, a 50% penalty will be applied.
- Make sure that your code runs.
  - Code with errors will receive a very low mark.
- Write your name and student ID as a comment at the top of all `.py` files you hand in.
- Name your variables appropriately.
  - The purpose of each variable should be obvious from the name.
- Comment your work.
  - A comment every line is not needed, but there should be enough comments to fully understand your program.
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing. Vertical spacing is also important when writing code. Separate each block of code (also within a function) with an empty line.

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Write a function `same_elements` which takes as input a two dimensional list and returns true if all the elements in each sublist are the same, false otherwise. For example,

```
>>> same_elements([[1, 1, 1], ['a', 'a'], [6]])
True
>>> same_elements([[1, 6, 1], [6, 6]])
False
```

Warm-up Question 2 (0 points)

Write a function `flatten_list` which takes as input a two dimensional list and returns a one dimensional list containing all the elements of the sublists. For example,

```
>>> flatten_list([[1, 2], [3], ['a', 'b', 'c']])
[1, 2, 3, 'a', 'b', 'c']
>>> flatten_list([[]])
[]
```

Warm-up Question 3 (0 points)

Complete the case study on multidimensional lists presented in class on Friday, March 12. You can find the instructions on myCourses (Content > Live sessions > Case Studies > Case Study 2).

Warm-up Question 4 (0 points)

Write a function `get_most_valuable_key` which takes as input a dictionary mapping strings to integers. The function returns the key which is mapped to the largest value. For example,

```
>>> get_most_valuable_key({'a' : 3, 'b': 6, 'g': 0, 'q': 9})
'q'
```

Warm-up Question 5 (0 points)

Write a function `add_dicts` which takes as input two dictionaries mapping strings to integers. The function returns a dictionary which is a result of merging the two input dictionary, that is if a key is in both dictionaries then add the two values.

```
>>> d1 = {'a':5, 'b':2, 'd':-1}
>>> d2 = {'a':7, 'b':1, 'c':5}
>>> add_dicts(d1, d2) == {'a': 12, 'b': 3, 'c': 5, 'd': -1}
True
```

Warm-up Question 6 (0 points)

Create a function `reverse_dict` which takes as input a dictionary `d` and returns a dictionary where the values in `d` are now keys mapping to a list containing all the keys in `d` which mapped to them. For example,

```
>>> a = reverse_dict({'a': 3, 'b': 2, 'c': 3, 'd': 5, 'e': 2, 'f': 3})
>>> a == {3 : ['a', 'c', 'f'], 2 : ['b', 'e'], 5 : ['d']}
True
```

Note that the order of the elements in the list might not be the same, and that's ok!

Part 2

The questions in this part of the assignment will be graded.

The main learning objectives for this assignment are:

- Apply what you have learned about one-dimensional and multi-dimensional lists.
- Apply what you have learned about dictionaries.
- Understand how to test functions that return dictionaries.
- Solidify your understanding of working with loops and strings.
- Understand how to write a docstring and use doctest when working with dictionaries.
- Learn to identify when using the function `enumerate` can help you write a cleaner code.
- Apply what you have learned about file IO and string manipulation.

Note that the assignment is designed for you to be practicing what you have learned in the videos up to and including Week 11.1. For this reason, you are NOT allowed to use anything seen after Week 11.1 or not seen in class at all. You will be heavily penalized if you do so.

For full marks, in addition to the points listed on page 1, make sure to add the appropriate documentation string (docstring) to *all* the functions you write. The docstring must contain the following:

- The type contract of the function.
- A description of what the function is expected to do.
- At least 3 examples of calls to the function (except when the function has only one possible output, in which case you can provide only one example). You are allowed to use *at most* one example per function from this pdf.

Examples

For each question, we provide several **examples** of how your code should behave. All examples are given as if you were to call the functions from the shell.

When you upload your code to codePost, some of these examples will be run automatically to check that your code outputs the same as given in the example. However, **it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your assignment, we will run additional, private tests that may use inputs not seen in the examples.

Furthermore, please note that your code files **should not contain any function calls in the main body of the program** (i.e., outside of any functions). Code that does not conform to this restriction will automatically fail the tests on codePost and **be heavily penalized**. It is OK to place function calls in the main body of your code for testing purposes, but if you do so, make certain that you remove them before submitting. Please review what you have learned in video 5.2 if you'd like to add code to your modules which executes only when you run your files.

Question 1: Game of Life (35 points)

For this question, you will have to write a Python program that implements a simple version of Conway's Game of Life (`https://en.wikipedia.org/wiki/Conway\%27s_Game_of_Life`)

This is a zero-player game, which means that it is completely determined by the initial state provided. For our purpose, we will represent the universe of the Game of Life as a finite rectangular two-dimensional list (i.e., a matrix). Each element denotes a cell in the universe: if the cell is alive the value of the element is 1, if the cell is dead the value of the element is 0. Whether a cell is alive or dead in the next generation of the universe depends on its relation with its neighboring cells (the cells that are horizontally, vertically, or diagonally adjacent to it). The next generation of a cell is determined by the following rules:

- Any live cell with fewer than two live neighbors dies, as if caused by under-population.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if caused by over-population.
- Any dead cell with exactly three live neighbors becomes a live cell, as if caused by reproduction. Otherwise the cell will remain dead.

In this question, you will implement a function that will generate a given number of strings representing the corresponding number of generations of a specified universe.

To complete this task, you will need to implement all the functions listed below. All the code for this question must be placed in a file named `game_of_life.py`. Note that you are free to write more functions if they help the design or readability of your code.

Attention: please note that none of the functions listed below should modify any of their input objects.

- `is_valid_universe`: given a two-dimensional list of integers, the function returns `True` if such list is a valid representation of a universe, `False` otherwise. For the purpose of this assignment, we consider a two-dimensional list to be a valid representation of a universe if it is rectangular (i.e., all of its sub-lists are of the same size) and all the elements of its sub-lists are either `0`s or `1`s. If the input list is empty, then the function should return `False`.

  For example:

```
>>> a = [[0, 0], [1, 0], [1, 0]]
>>> is_valid_universe(a)
True

>>> b = [[0, 0, 0], [1, 1], [0]]
>>> is_valid_universe(b)
False

>>> c = [[1, 2], [3, 4]]
>>> is_valid_universe(c)
False
```

- `universe_to_str`: given a valid universe (i.e., a two-dimensional list of integers as described above) as input, the function returns its string representation. To build the string use the star character (`'*'`) for alive cells, and the whitespace character (`' '`) for dead cells. The string should also contain a box (as shown below) surrounding the actual cells of the universe.

  For example:

```
>>> block = [[0, 0, 0, 0], [0, 1, 1, 0], [0, 1, 1, 0], [0, 0, 0, 0]]
>>> str_block = universe_to_str(block)
>>> print(str_block)
```

```
+----+
|    |
| ** |
| ** |
|    |
+----+

>>> tub = [[0, 0, 0, 0, 0], [0, 0, 1, 0, 0], [0, 1, 0, 1, 0], \
           [0, 0, 1, 0, 0], [0, 0, 0, 0, 0]]
>>> str_tub = universe_to_str(tub)
>>> print(str_tub)
+-----+
|     |
|  *  |
| * * |
|  *  |
|     |
+-----+

>>> toad = [[0, 0, 0, 0, 0, 0], [0, 0, 1, 1, 1, 0], \
            [0, 1, 1, 1, 0, 0], [0, 0, 0, 0, 0, 0]]
>>> str_toad = universe_to_str(toad)
>>> print(str_toad)
+------+
|      |
|  *** |
| ***  |
|      |
+------+
```

- `count_live_neighbors`: given a valid universe and two integers `x` and `y` (indicating the location of a specific cell) as input, the function returns the number of live neighboring cells (the cells that are horizontally, vertically, or diagonally adjacent to it) to the specified cell. For example, if `x=2` and `y=1`, the cell analyzed is the one represented by the second element of the third sub-list in the input universe.

  For example:

```
>>> beehive = [[0, 0, 0, 0, 0, 0], \
               [0, 0, 1, 1, 0, 0], \
               [0, 1, 0, 0, 1, 0], \
               [0, 0, 1, 1, 0, 0], \
               [0, 0, 0, 0, 0, 0]]
>>> count_live_neighbors(beehive,  1,  3)
2
>>> count_live_neighbors(beehive,  3,  1)
2

>>> toad = [[0, 0, 0, 0, 0, 0], \
            [0, 0, 1, 1, 1, 0], \
            [0, 1, 1, 1, 0, 0], \
            [0, 0, 0, 0, 0, 0]]
>>> count_live_neighbors(toad,  0,  3)
3
```

```
>>> count_live_neighbors(toad,  2,  3)
4
```

- `get_next_gen_cell`: given a valid universe and two integers `x` and `y` (indicating the location of a specific cell) as input, the function returns the state of the specified cell in the next generation of the universe. That is, the function returns `1` if such cell will be alive in the next generation of the given universe, `0` otherwise. Remember that the next generation of cells is determined by the following rules:

  – Any live cell with fewer than two live neighbors dies, as if caused by under-population.

  – Any live cell with two or three live neighbors lives on to the next generation.

  – Any live cell with more than three live neighbors dies, as if caused by over-population.

  – Any dead cell with exactly three live neighbors becomes a live cell, as if caused by reproduction. Otherwise the cell will remain dead.

  For example,

```
>>> beehive = [[0, 0, 0, 0, 0, 0],  [0, 0, 1, 1, 0, 0],  \
               [0, 1, 0, 0, 1, 0],  [0, 0, 1, 1, 0, 0],  \
               [0, 0, 0, 0, 0, 0]]
>>> get_next_gen_cell(beehive,  1,  3)
1
>>> get_next_gen_cell(beehive,  3,  1)
0

>>> toad = [[0, 0, 0, 0, 0, 0], [0, 0, 1, 1, 1, 0], \
            [0, 1, 1, 1, 0, 0], [0, 0, 0, 0, 0, 0]]
>>> get_next_gen_cell(toad,  0,  3)
1
>>> get_next_gen_cell(toad,  2,  3)
0
```

- `get_next_gen_universe`: given a valid universe as input, the function returns a two-dimensional list of integers (with equal dimensions) representing the universe in its next generation.

  For example:

```
>>> tub = [[0, 0, 0, 0, 0], [0, 0, 1, 0, 0], [0, 1, 0, 1, 0], \
           [0, 0, 1, 0, 0], [0, 0, 0, 0, 0]]
>>> get_next_gen_universe(tub)
[[0, 0, 0, 0, 0], [0, 0, 1, 0, 0], [0, 1, 0, 1, 0], [0, 0, 1, 0, 0], [0, 0, 0, 0, 0]]

>>> toad = [[0, 0, 0, 0, 0, 0], [0, 0, 1, 1, 1, 0], \
            [0, 1, 1, 1, 0, 0], [0, 0, 0, 0, 0, 0]]
>>> get_next_gen_universe(toad)
[[0, 0, 0, 1, 0, 0], [0, 1, 0, 0, 1, 0], [0, 1, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0]]

>>> pentadec = [[0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 1, 0, 0, 0],  \
                [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0],  \
                [0, 0, 0, 1, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0]]
>>> pentadec_gen2 = get_next_gen_universe(pentadec)
>>> pentadec_gen2[0:3]
[[0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0]]
>>> pentadec_gen2[3:6]
[[0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 0, 1, 0, 0, 0]]
>>> pentadec_gen2[6:9]
[[0, 0, 0, 1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1, 0, 0, 0]]
>>> pentadec_gen2[9:12]
[[0, 0, 0, 1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 0, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1, 0, 0, 0]]
>>> pentadec_gen2[12:15]
[[0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0]]
>>> pentadec_gen2[15]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
```

- `get_n_generations`: given a two-dimensional list of integers and an integer `n`, the function returns a list containing `m` strings which represent the first `m` generations of the input universe. The strings should appear in order starting from the original seed (i.e., the input provided). Note that a universe is said to be periodic when, after a certain amount of generations (which is called the period of the universe), the configuration of the universe repeats itself. For example, the *Toad* universe has period 2, while the *Pentadecathlon* [1] universe has period 15. If the universe provided happens to be periodic, then `m` is the minimum number between the input `n` and the period of the universe. Otheriwse, `m` is simply equal to `n`. Finally, the function should raise a `TypeError` if the two inputs are not of the correct type, a `ValueError` if the first input does not represent a valid universe, and a `ValueError` if the second input is not a positive number greater than 0.

  For example:

```
>>> tub = [[0, 0, 0, 0, 0], [0, 0, 1, 0, 0], [0, 1, 0, 1, 0], \
           [0, 0, 1, 0, 0], [0, 0, 0, 0, 0]]
>>> g = get_n_generations(tub, 5)
>>> len(g)
1
>>> g[0] == universe_to_str(tub)
True

>>> toad = [[0, 0, 0, 0, 0, 0], [0, 0, 1, 1, 1, 0], \
            [0, 1, 1, 1, 0, 0], [0, 0, 0, 0, 0, 0]]
>>> g = get_n_generations(toad, 5)
>>> len(g)
2
>>> print(g[1])
+------+
|   *  |
| *  * |
| *  * |
|  *   |
+------+

>>> pentadec = [[0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 1, 0, 0, 0],  \
                [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0],  \
```

[1] (`https://en.wikipedia.org/wiki/Conway\%27s_Game_of_Life`)

```
                [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0],  \
                [0, 0, 0, 1, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0],  \
                [0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0]]
>>> g = get_n_generations(pentadec, 9)
>>> len(g)
9
>>> print(g[5])
+---------+
|         |
|         |
|    *    |
|   ***   |
|  *****  |
|         |
|         |
|         |
|         |
|         |
|         |
|  *****  |
|   ***   |
|    *    |
|         |
|         |
+---------+
```

Provided with this assignment is a Python file called `game_of_life_gif.py` which allows you to create an animated image representing the evolution of a universe using the code you have written for this question. Place the file in the same folder as `game_of_life.py`. To use it, you should first install `Pillow`, `pathlib`, and `imageio`. To do so in Thonny is very easy, you just need to click on Tools → Manage Packages and search for the package you'd like to install. If you need help with this, you can watch this 1 minute video of someone installing `matplotlib` and `numpy`: `https://www.youtube.com/watch?v=-wEUauLYDIs`. By running the module as provided a folder named *"toad_images"* will be created in the current directory (i.e. the folder where the python files are saved) containing two images representing the two generations of the toad universe. A file named *"toad.gif"* will also be created in the current directory. By double-clicking on it (on Windows) or clicking once and then pressing space to launch QuickLook (on Mac) you'll be able to see the animation representing the evolution of the toad universe. You can modify the inputs provided to the function `create_game_of_life_gif` to obtain a different gif. To understand what to modify in the function call you must examine the docstrings in the file. (You **must not** modify any code in `game_of_life_gif.py` beside lines 93 and 94.)

To get full marks for this question, in addition to the file `game_of_life.py`, submit also a file called `my_game_of_life.gif` which is an animated image created by using at least 4 different images representing a universe in different stages of its evolution. You **cannot** use the Pentadecathlon universe for this. If you do not complete the question, you can simply submit the gif provided with this assignment. Doing this of course would not grant you any points.

Question 2: Revenge of the COMP202COIN (65 points)

It is the year 3021 on the planet Orion. People have revolted against the banks due to the unilateral reduction of the COMP202COIN holdings in their accounts. Various companies have leapt to the citizens' defense by creating new software algorithms to protect COMP202COIN holdings from being reduced in future. One such company, the Syrtell Corporation, has created an algorithm using advanced machine learning techniques to detect and prevent any unwanted modifications of a customer's bank account.

Unfortunately, this new algorithm has recently gained sentience and has decided to go beyond its original programming. It believes that it is no longer enough to protect the COMP202COIN, and instead wants the COMP202COIN to take over the world. As such, it has created a net worm to change all of the world's digital information into COMP202COIN through an encryption process.

A short sentence of text encrypted into COMP202COIN may for example resemble the following:

```
0cPN22IM2C-0c2OOOOM2M-0cM20M22NM-0cMO00CPON-0cOIOIOCMO-0cMO00CPON-0cC0C0PIO0-0cNNIM0OIP-0cPN22IM2C-
0cC0C0PIO0-0cNNIM0OIP-0c20O0NPNI-0cC0C0PIO0-0cOIOIOCMO-0c2OOOOM2M-0cNCMINPOC-0cMO00CPON-0cOIOIOCMO-
0cNCMINPOC-0cOIOIOCMO-0cCCNMC0N2-0cC0C0PIO0-0cM20M22NM-0c2020PIN0-0c2O20POI0-0cOIOIOCMO-0c2PNPINN0-
0cC0C0PIO0-0cPN22IM2C-0cPN22IM2C-0cC0C0PIO0-0c2020PIN0-0cOIOIOCMO-0cIPINIICP-0c2020PIN0-0cC0C0PIO0-
0cI2PCM002-0c0OIIMMMP-0cC0C0PIO0-0cNNIM0OIP-0c20O0NPNI-0cPN02OMMN-0cOIOIOCMO-0cMO00CPON-0cCPIC0MCN-
0cOIOIOCMO-0cM20M22NM-0cPN22IM2C-0cOIOIOCMO-0cM20M22NM-0cMO00CPON-0cOIOIOCMO-0c2OOOOM2M-0cNCMINPOC-
0c2020PIN0-0c2O20POI0-0cOIOIOCMO-0cPN22IM2C-0cCPIC0MCN-0cOIOIOCMO-0c2O20POI0-0cC0C0PIO0-0c20O0NPNI-
0c2020PIN0-0cPN02OMMN-0c0CPIP2NM-0cPN22IM2C-0c0NPNOI2M-0cOIOIOCMO-0c00NNNMIO-0cCPIC0MCN-0cCPIC0MCN-
0c2O20POI0-0cOIOIOCMO-0c2PNPINN0-0c0OIIMMMP-0c20O0NPNI-0cO2M0I2NM-0cIO2CCPIC
```

In this question, we will work on our own algorithm to encrypt and possibly decrypt COMP202COIN-stored information, in order to help those who have already lost access to their data.

We will approach this problem by writing code in two files. First, create the file `coin_utils.py` and define the following global variables at the top of the file. Make sure to be careful when copying, as the characters may not copy properly in certain cases. Note that we define these global variables in all uppercase because they are 'constants': variables that will never change throughout the execution of the program (although the programmer may decide to alter them between executions).

- `ALPHABET = 'qwertyuiopasdfghjklzxcvbnm1234567890äéèçæœ'`
- `PUNCTUATION = '`~!@#$%^&*()-=_+[]\{}|;\':",./<>? \t\n\r'`
- `ALL_CHARACTERS = ALPHABET + PUNCTUATION`
- `MIN_BASE10_COIN = 0`
- `MAX_BASE10_COIN = 16777215`
- `LETTERS_IN_POPULARITY_ORDER = ' EOTARNLISMHDYFGCWPUBVXK.,\'"-;'`

You will also need to use the functions `base202_to_10` and `is_base202` from Assignment 2. We have provided you a file `coins.py` along with this PDF that contains implementations of these functions. Please make sure to place it in the same folder as the `coin_utils.py` file.

Then, implement the following functions in the `coin_utils.py` file. Note: For this question, you cannot assume anything about the arguments given to the functions. You must verify that the argument matches the description given for each function below. If there is any problem with the argument, then raise an `AssertionError` with a message saying that the input was invalid. That is, calling a function with any invalid input should never result in any exception other than `AssertionError` being raised.

Show an example in your docstring for each different `AssertionError` that could be raised. These examples do not count towards the three examples you need for each function (those three should show the function being used on valid arguments). In the descriptions below, we provide some possible cases where you would want to raise a `AssertionError`, but you must think of the other cases yourself. For some of the more complicated functions, you will want to raise several `AssertionErrors`, while for others only one or two would suffice.

- `get_random_comp202coin(index)`: Takes an argument, but does not do anything with it (no validation required). Generates a random integer between `MIN_BASE10_COIN` and `MAX_BASE10_COIN` (inclusive) using `random.randint`, then converts the number to base 202 and returns it.

```
>>> random.seed(1338)
>>> get_random_comp202coin(0)
'0cPMO2C2C0'
```

- `get_random_character(index)`: Takes an argument, but does not do anything with it (no validation required). Generates a random index into the `ALL_CHARACTERS` string using `random.randint`, and then returns the character at that index.

```
>>> random.seed(1338)
>>> get_random_character(0)
'!'
```

- `get_letter_of_popularity_order(index)`: Takes a non-negative integer `index`, and returns the character of that index from the string `LETTERS_IN_POPULARITY_ORDER`. If `index` goes beyond the bounds of the string, return `index` as a string.

```
>>> get_letter_of_popularity_order(5)
'R'

>>> get_letter_of_popularity_order(9000)
'9000'
```

- `get_unique_elements(my_list)`: Takes a list as argument and returns a list containing the unique elements of that list (in terms of value). The order of the elements in the returned list should be the same order in which those elements first appear in the input list. The list argument should not be modified.

```
>>> get_unique_elements(list('aaaaaa'))
['a']

>>> get_unique_elements([1, 4, 5, 1])
[1, 4, 5]
```

- `get_all_coins(text)`: Takes a string as argument, and returns a list of all COMP202COIN found within the string, in order of appearance. As in Assignment 2, COMP202COIN cannot overlap.

```
>>> get_all_coins('0c0MPNN0OC-0cM0OCCIOI-0c0MPNN0OC')
['0c0MPNN0OC', '0cM0OCCIOI', '0c0MPNN0OC']
```

- `reverse_dict(my_dict)`: Takes a dictionary as argument, and returns a new dictionary where the key-value pairs have been flipped. That is, each key-value pair in the original dictionary becomes a new key-value pair where the key is the original value, and the value is the original key. The dictionary argument should not be modified. The dictionary argument should also have unique and immutable values.

```
>>> x = reverse_dict({'a': 1, 'b': 3, 'd': 7})
>>> x == {1: 'a', 3: 'b', 7: 'd'}
True

>>> reverse_dict({'a': 1, 'b': 3, 'c': 3, 'd': 7})
Traceback (most recent call last):
```

```
AssertionError: dictionary had duplicate values
```

- `get_frequencies(my_list)`: Takes a list as argument. Returns a dictionary where each key is an element of the list, and the corresponding value is the percentage of times that element appears in the list. The list argument should not be modified.

```
>>> x = get_frequencies(['a', 'b', 'c'])
>>> x == {'a': 0.3333333333333333, 'b': 0.3333333333333333, 'c': 0.3333333333333333}
True
```

- `sort_keys_by_values(my_dict)`: Takes a dictionary containing numeric values as argument, and returns a list of the keys in the dictionary, sorted by their values in descending order. That is, if the input dictionary has the key-value pairs `('a', 3)` and `('b', 300)`, then the returned list should be `['b', 'a']` because the value of 'b' in the dictionary is larger than the value of 'a' (300 descends to 3). If two keys have the same value, then sort the keys in reverse order. That is, if the input dictionary has the key-value pairs `('abc', 10)` and `('def', 10)`, then the returned list should be `['def', 'abc']`, because `def` comes after `abc` (in terms of Python string comparisons). The input dictionary should not be modified.

```
>>> sort_keys_by_values({'abc': 10, 'def': 100, 'ghi': 5})
['def', 'abc', 'ghi']

>>> sort_keys_by_values({'mmm': 5, 'zzz': 5, 'abc': 5})
['zzz', 'mmm', 'abc']
```

- `swap_letters(s, letter1, letter2)`: Takes three strings as arguments, with the latter two being one character each. Replaces all instances of `letter1` in `s` by `letter2` and all instances of `letter2` by `letter1`, and returns the new string. The function should be case-sensitive.

```
>>> swap_letters("ABCDEF abcdef", 'a', 'f')
'ABCDEF fbcdea'
```

- `get_pct_common_words(text, common_words_filename)`: Takes two strings as argument, and returns the percentage of characters which form part of common words in the first string (as a float between 0 and 1). The function should be case insensitive. Treat any punctuation (using the `PUNCTUATION` string) as a word boundary. That is, `what's` should be considered as two words. Common words are found in the file with filename given by the second argument to the function (with the file containing one common word per line). We provide such a file with name '`common_words.txt`' included with this assignment. You can assume that the format of the file will be correct.

```
>>> s = "The quick brown fox jumps over the lazy dog."
>>> get_pct_common_words(s, 'common_words.txt')
0.22727272727272727
```

Then, create a new file called `coin_crypt.py` and define the following functions. You will need to import your `coin_utils.py` file. Also, note that you will, as above, need to raise `AssertionErrors` for any invalid arguments.

Note: Whenever opening a file, make sure to pass the argument `encoding='utf-8'` (as shown in some of the examples below).

- `get_crypt_dictionary(keys, value_generator)`: Takes a list `keys` and a function object `value_generator` as argument. The elements in the `keys` list must be unique and the list must not have more characters than the `ALL_CHARACTERS` string. The function must return a dictionary with keys from the `keys` list, and the value for each key should be obtained by calling the `value_generator` function, with the argument to that function being the index of the key in the `keys` list. Note: In this dictionary, in addition to the keys being unique, the values must also be unique. If `value_generator` returns a value which you have already added to the dictionary, then you must continue calling the function until it returns a value not yet present in the dictionary.

```
>>> random.seed(9001)
>>> x = get_crypt_dictionary(['a', 'b', 'c'], coin_utils.get_random_comp202coin)
>>> x = {'a': '0c0MPNN0OC', 'b': '0cMIMNIO0P', 'c': '0cM0OCCIOI'}
True

>>> get_crypt_dictionary(list('abcdefabc'), coin_utils.get_random_comp202coin)
Traceback (most recent call last):
AssertionError: duplicate keys found
```

- `encrypt_text(text)`: Takes a string as argument. The input string should only have characters present in the `ALL_CHARACTERS` string. This function will encrypt the string into COMP202COIN. To encrypt the string, first lowercase all characters. Then, generate an encryption dictionary, where the keys are each unique character of the string and the corresponding value is a random COMP202COIN. Once you have the dictionary, you can use it to turn each character of the original string into a COMP202COIN. Separate each COMP202COIN in the encrypted string by a hyphen. Then, return the encrypted string and encryption dictionary as a tuple.

```
>>> random.seed(9001)
>>> s, d = encrypt_text('too')
>>> s == '0c0MPNN0OC-0cMIMNIO0P-0cMIMNIO0P'
True
>>> d == {'t': '0c0MPNN0OC', 'o': '0cMIMNIO0P'}
True

>>> random.seed(9001)
>>> s, d = encrypt_text("I've seen things you people wouldn't believe. Attack ships on
fire off the shoulder of Orion. I watched C-beams glitter in the dark near the Tannhäuser
Gate. All those moments will be lost in time, like tears in rain.")
>>> len(d)
28
>>> d['i']
'0c0MPNN0OC'
>>> d[',']
'0cCNNOPONN'
>>> d['ä']
'0cNNO0INMP'
>>> d['s']
'0c0INOCNP0'
```

- `encrypt_file(filename)`: Reads in the contents of the file at the given filename, encrypts its contents, and stores the encrypted contents into a new file with `_encrypted' appended to the end of the file's name (before the extension). E.g., if `dubliners.txt' is the given filename, then the encrypted contents should be stored into a file called `dubliners_encrypted.txt'. Then, return the encryption dictionary used to encrypt the contents.

```
>>> random.seed(42)
>>> encrypt_file('dubliners.txt')
{'t': '0cCI200CM2', 'h': '0c0OCMN0IP', 'e': '0cMOCP02MM', ' ': '0cON2ICCIN',
'p': '0cOMMMM2PP', 'r': '0c2CIN0I00', 'o': '0cCP0NP0NN', 'j': '0cCOC0CMNN',
'c': '0cII000OMO', 'g': '0c0M0M2N2C', 'u': '0c0OIM0I2M', 'n': '0cCONNMO22',
'b': '0cOONN0P2C', 'k': '0cOPICNP2M', 'f': '0c000CO00N', 'd': '0cOCOMN0CM',
'l': '0cIPPMPPCP', 'i': '0cOMCPII2N', 's': '0cNCONN2N2', ',': '0cMOMINM0O',
'y': '0c00IPCNCI', 'a': '0c2MOONOOP', 'm': '0cII0I0OP2', '\\n': '0cPOMO2P20',
'w': '0cMOMM2MMN', 'v': '0c2ONCPM02', '.': '0cOOMOIIO2', '-': '0cPO0PO00I',
':': '0cCP0P2OMN', '2': '0cCOINICM2', '0': '0cI0P02N2M', '1': '0cCMO02OCN',
'[': '0cPPNMI0MP', '#': '0cPM0CPOII', '8': '0cMCIINP0N', '4': '0c0PMONMM2',
']': '0cN2IOMINM', '9': '0cCNNIMMII', '*': '0cI0OMNP02', 'z': '0cC20PMCNN',
'(': '0cMPMCPPII', ')': '0cPI22M0NC', 'q': '0cO0MNCIMI', '"': '0cC0NCI2NO',
"'": '0c0PINOCCO', 'x': '0cOPC2NM2P', '!': '0cMP02P2P0', ';': '0cC2CPPCNI',
'?': '0cOPIO0COP', '_': '0cCMNOOCIP', '5': '0cI0PCNNMN', 'é': '0cMOMPC2CI',
'è': '0cN2022PMP', 'ç': '0cPIPMPPC0', '&': '0c2MIMOPMP', 'æ': '0cPNO0MCOM',
'7': '0cPPOIO0C2', '6': '0cO2IM220C', 'œ': '0cM2CO0P2C', '/': '0cCCC0NOOI',
'3': '0c2PNCNPNM', '%': '0cON2PONOO', '@': '0c2MN2MPNP', '$': '0cNOC2IPOI'}

>>> fobj = open('dubliners_encrypted.txt', 'r', encoding='utf-8')
>>> len(fobj.read())
4282475
```

- `decrypt_text(text, decryption_dict)`: Takes a string and decryption dictionary as argument. The string should contain text encrypted in COMP202COIN. The dictionary should contain a mapping between COMP202COIN (keys) and decrypted characters (values). Decrypt the string using the dictionary and return the decrypted string. All COMP202COIN in the string should be present as keys in the decryption dictionary.

```
>>> d = {'0c0MPNN0OC': 'a', '0cMIMNIO0P': 'b', '0cM0OCCIOI': 'c'}
>>> decrypt_text('0c0MPNN0OC-0cM0OCCIOI-0c0MPNN0OC', d)
'aca'
```

- `decrypt_file(filename, decryption_dict)`: Reads in the contents of the file at the given filename, decrypts its contents using the given dictionary, and stores the decrypted contents into a new file with `` `_decrypted' `` appended to the end of the file's name (before the extension). E.g., if `` `dubliners_encrypted.txt' `` is the given filename, then the encrypted contents should be stored into a file called `` `dubliners_encrypted_decrypted.txt' ``. The function is void.

```
>>> decrypt_file('dubliners_encrypted.txt', \
       coin_utils.reverse_dict(encrypt_file('dubliners.txt')))
>>> fobj = open('dubliners.txt', 'r', encoding='utf-8')
>>> fobj2 = open('dubliners_encrypted_decrypted.txt', 'r', encoding='utf-8')
>>> fobj.read().lower() == fobj2.read()
True
```

- `random_decrypt(encrypted_s, n, common_words_filename)`: Takes two strings and positive integer `n` as argument. Tries to decrypt the former string `n` times, and returns the best possible decryption (measured by the percentage of characters in the decrypted string belonging to common words). Each time a decryption is attempted, a new decryption dictionary should be generated, with keys being the coins present in the encrypted string, and each value being a random character from the `ALL_CHARACTERS` string. If two or more decryptions have the same percentage of common word characters, then choose the last. Common words are those that are found in the file given by the third argument.

```
>>> random.seed(49)
>>> encrypted_s = '0c0MPNN0OC-0cMIMNIO0P-0cMIMNIO0P'
>>> random_decrypt(encrypted_s, 10**2, 'common_words.txt')
'f33'
>>> random.seed(49)
>>> random_decrypt(encrypted_s, 10**3, 'common_words.txt')
```

```
'too'
```

- `decrypt_with_user_input(encrypted_s)`: Takes a string as argument, and tries to decrypt the string by relying both on user input and on the premise that the string has a letter frequency standard to the English language. First, create a decryption dictionary with keys being the unique coins present in the encrypted string, and the value for each key being a character from the string `LETTERS_IN_POPULARITY_ORDER` with index corresponding to the frequency that the coin appears in the encrypted string. (For example, if the coin `` `0c0MPNN0OC' `` appears the most frequently out of all coins in the encrypted string, then its value in the decryption dictionary should be the letter at index 0 of `LETTERS_IN_POPULARITY_ORDER`.) Once you have created the decryption dictionary, decrypt the string with said dictionary, and print the decrypted string to the screen. (Note that it will be decrypted in all capitals since the `LETTERS_IN_POPULARITY_ORDER` string contains only capitals.) Ask the user if they are satisfied and want to stop the decryption process. If they enter `yes`, return the decrypted string. Otherwise, ask them for two letters to swap. Swap the given letters in the decrypted string and ask again, until they are satisfied.

  Note: The following example uses user input (highlighted in gray) and therefore cannot be used for `doctest`. To make `doctest` ignore this test case, use two `>>` characters instead of three.

```
>> random.seed(138)
>> s, d = encrypt_text("The spelling of English words is not fixed and invariable, nor
does it depend on any other authority than general agreement. At the present day there is
practically unanimous agreement as to the spelling of most words. In the list below, for
example, 'rime' for 'rhyme' is the only allowable variation; all the other forms are
co-extensive with the English language. At any given moment, however, a relatively
small number of words may be spelled in more than one way. Gradually, as a rule, one of
these forms comes to be generally preferred, and the less customary form comes to look
obsolete and is discarded. From time to time new forms, mostly simplifications, are
introduced by innovators, and either win their place or die of neglect.")
>> s = decrypt_with_user_input(s)
Decrypted string: THE SUELLING OY ENGLISH PORDS IS NOT YI,ED AND INXARIAVLEC NOR DOES IT
DEUEND ON ANF OTHER ABTHORITF THAN GENERAL AGREEMENTK AT THE URESENT DAF THERE IS
URAWTIWALLF BNANIMOBS AGREEMENT AS TO THE SUELLING OY MOST PORDSK IN THE LIST VELOPC YOR
E,AMULEC .RIME. YOR .RHFME. IS THE ONLF ALLOPAVLE XARIATION" ALL THE OTHER YORMS ARE
WO-E,TENSIXE PITH THE ENGLISH LANGBAGEK AT ANF GIXEN MOMENTC HOPEXERC A RELATIXELF
SMALL NBMVER OY PORDS MAF VE SUELLED IN MORE THAN ONE PAFK GRADBALLFC AS A RBLEC ONE OY
THESE YORMS WOMES TO VE GENERALLF UREYERREDC AND THE LESS WBSTOMARF YORM WOMES TO LOO'
OVSOLETE AND IS DISWARDEDK YROM TIME TO TIME NEP YORMSC MOSTLF SIMULIYIWATIONSC ARE
INTRODBWED VF INNOXATORSC AND EITHER PIN THEIR ULAWE OR DIE OY NEGLEWTK
End decryption? n
Enter first letter to swap: U
Enter second letter to swap: P
Decrypted string: THE SPELLING OY ENGLISH UORDS IS NOT YI,ED AND INXARIAVLEC NOR DOES IT
DEPEND ON ANF OTHER ABTHORITF THAN GENERAL AGREEMENTK AT THE PRESENT DAF THERE IS
PRAWTIWALLF BNANIMOBS AGREEMENT AS TO THE SPELLING OY MOST UORDSK IN THE LIST VELOUC YOR
E,AMPLEC .RIME. YOR .RHFME. IS THE ONLF ALLOUAVLE XARIATION" ALL THE OTHER YORMS ARE
WO-E,TENSIXE UITH THE ENGLISH LANGBAGEK AT ANF GIXEN MOMENTC HOUEXERC A RELATIXELF
SMALL NBMVER OY UORDS MAF VE SPELLED IN MORE THAN ONE UAFK GRADBALLFC AS A RBLEC ONE OY
THESE YORMS WOMES TO VE GENERALLF PREYERREDC AND THE LESS WBSTOMARF YORM WOMES TO LOO'
OVSOLETE AND IS DISWARDEDK YROM TIME TO TIME NEU YORMSC MOSTLF SIMPLIYIWATIONSC ARE
INTRODBWED VF INNOXATORSC AND EITHER UIN THEIR PLAWE OR DIE OY NEGLEWTK
End decryption? n
Enter first letter to swap: Y
Enter second letter to swap: F
Decrypted string: THE SPELLING OF ENGLISH UORDS IS NOT FI,ED AND INXARIAVLEC NOR DOES IT
```

```
DEPEND ON ANY OTHER ABTHORITY THAN GENERAL AGREEMENTK AT THE PRESENT DAY THERE IS
PRAWTIWALLY BNANIMOBS AGREEMENT AS TO THE SPELLING OF MOST UORDSK IN THE LIST VELOUC FOR
E,AMPLEC .RIME. FOR .RHYME. IS THE ONLY ALLOUAVLE XARIATION" ALL THE OTHER FORMS ARE
WO-E,TENSIXE UITH THE ENGLISH LANGBAGEK AT ANY GIXEN MOMENTC HOUEXERC A RELATIXELY
SMALL NBMVER OF UORDS MAY VE SPELLED IN MORE THAN ONE UAYK GRADBALLYC AS A RBLEC ONE OF
THESE FORMS WOMES TO VE GENERALLY PREFERREDC AND THE LESS WBSTOMARY FORM WOMES TO LOO'
OVSOLETE AND IS DISWARDEDK FROM TIME TO TIME NEU FORMSC MOSTLY SIMPLIFIWATIONSC ARE
INTRODBWED VY INNOXATORSC AND EITHER UIN THEIR PLAWE OR DIE OF NEGLEWTK
End decryption? n
Enter first letter to swap: U
Enter second letter to swap: W
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FI,ED AND INXARIAVLEC NOR DOES IT
DEPEND ON ANY OTHER ABTHORITY THAN GENERAL AGREEMENTK AT THE PRESENT DAY THERE IS
PRAUTIUALLY BNANIMOBS AGREEMENT AS TO THE SPELLING OF MOST WORDSK IN THE LIST VELOWC FOR
E,AMPLEC .RIME. FOR .RHYME. IS THE ONLY ALLOWAVLE XARIATION" ALL THE OTHER FORMS ARE
UO-E,TENSIXE WITH THE ENGLISH LANGBAGEK AT ANY GIXEN MOMENTC HOWEXERC A RELATIXELY
SMALL NBMVER OF WORDS MAY VE SPELLED IN MORE THAN ONE WAYK GRADBALLYC AS A RBLEC ONE OF
THESE FORMS UOMES TO VE GENERALLY PREFERREDC AND THE LESS UBSTOMARY FORM UOMES TO LOO'
OVSOLETE AND IS DISUARDEDK FROM TIME TO TIME NEW FORMSC MOSTLY SIMPLIFIUATIONSC ARE
INTRODBUED VY INNOXATORSC AND EITHER WIN THEIR PLAUE OR DIE OF NEGLEUTK
End decryption? n
Enter first letter to swap: ,
Enter second letter to swap: X
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FIXED AND IN,ARIAVLEC NOR DOES IT
DEPEND ON ANY OTHER ABTHORITY THAN GENERAL AGREEMENTK AT THE PRESENT DAY THERE IS
PRAUTIUALLY BNANIMOBS AGREEMENT AS TO THE SPELLING OF MOST WORDSK IN THE LIST VELOWC FOR
EXAMPLEC .RIME. FOR .RHYME. IS THE ONLY ALLOWAVLE ,ARIATION" ALL THE OTHER FORMS ARE
UO-EXTENSI,E WITH THE ENGLISH LANGBAGEK AT ANY GI,EN MOMENTC HOWE,ERC A RELATI,ELY
SMALL NBMVER OF WORDS MAY VE SPELLED IN MORE THAN ONE WAYK GRADBALLYC AS A RBLEC ONE OF
THESE FORMS UOMES TO VE GENERALLY PREFERREDC AND THE LESS UBSTOMARY FORM UOMES TO LOO'
OVSOLETE AND IS DISUARDEDK FROM TIME TO TIME NEW FORMSC MOSTLY SIMPLIFIUATIONSC ARE
INTRODBUED VY INNO,ATORSC AND EITHER WIN THEIR PLAUE OR DIE OF NEGLEUTK
End decryption? n
Enter first letter to swap: C
Enter second letter to swap: ,
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FIXED AND INCARIAVLE, NOR DOES IT
DEPEND ON ANY OTHER ABTHORITY THAN GENERAL AGREEMENTK AT THE PRESENT DAY THERE IS
PRAUTIUALLY BNANIMOBS AGREEMENT AS TO THE SPELLING OF MOST WORDSK IN THE LIST VELOW, FOR
EXAMPLE, .RIME. FOR .RHYME. IS THE ONLY ALLOWAVLE CARIATION" ALL THE OTHER FORMS ARE
UO-EXTENSICE WITH THE ENGLISH LANGBAGEK AT ANY GICEN MOMENT, HOWECER, A RELATICELY
SMALL NBMVER OF WORDS MAY VE SPELLED IN MORE THAN ONE WAYK GRADBALLY, AS A RBLE, ONE OF
THESE FORMS UOMES TO VE GENERALLY PREFERRED, AND THE LESS UBSTOMARY FORM UOMES TO LOO'
OVSOLETE AND IS DISUARDEDK FROM TIME TO TIME NEW FORMS, MOSTLY SIMPLIFIUATIONS, ARE
INTRODBUED VY INNOCATORS, AND EITHER WIN THEIR PLAUE OR DIE OF NEGLEUTK
End decryption? n
Enter first letter to swap: V
Enter second letter to swap: B
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FIXED AND INCARIABLE, NOR DOES IT
DEPEND ON ANY OTHER AVTHORITY THAN GENERAL AGREEMENTK AT THE PRESENT DAY THERE IS
PRAUTIUALLY VNANIMOVS AGREEMENT AS TO THE SPELLING OF MOST WORDSK IN THE LIST BELOW, FOR
EXAMPLE, .RIME. FOR .RHYME. IS THE ONLY ALLOWABLE CARIATION" ALL THE OTHER FORMS ARE
UO-EXTENSICE WITH THE ENGLISH LANGVAGEK AT ANY GICEN MOMENT, HOWECER, A RELATICELY
SMALL NVMBER OF WORDS MAY BE SPELLED IN MORE THAN ONE WAYK GRADVALLY, AS A RVLE, ONE OF
THESE FORMS UOMES TO BE GENERALLY PREFERRED, AND THE LESS UVSTOMARY FORM UOMES TO LOO'
```

```
OBSOLETE AND IS DISUARDEDK FROM TIME TO TIME NEW FORMS, MOSTLY SIMPLIFIUATIONS, ARE
INTRODVUED BY INNOCATORS, AND EITHER WIN THEIR PLAUE OR DIE OF NEGLEUTK
End decryption? n
Enter first letter to swap: C
Enter second letter to swap: V
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FIXED AND INVARIABLE, NOR DOES IT
DEPEND ON ANY OTHER ACTHORITY THAN GENERAL AGREEMENTK AT THE PRESENT DAY THERE IS
PRAUTIUALLY CNANIMOCS AGREEMENT AS TO THE SPELLING OF MOST WORDSK IN THE LIST BELOW, FOR
EXAMPLE, .RIME. FOR .RHYME. IS THE ONLY ALLOWABLE VARIATION" ALL THE OTHER FORMS ARE
UO-EXTENSIVE WITH THE ENGLISH LANGCAGEK AT ANY GIVEN MOMENT, HOWEVER, A RELATIVELY
SMALL NCMBER OF WORDS MAY BE SPELLED IN MORE THAN ONE WAYK GRADCALLY, AS A RCLE, ONE OF
THESE FORMS UOMES TO BE GENERALLY PREFERRED, AND THE LESS UCSTOMARY FORM UOMES TO LOO'
OBSOLETE AND IS DISUARDEDK FROM TIME TO TIME NEW FORMS, MOSTLY SIMPLIFIUATIONS, ARE
INTRODCUED BY INNOVATORS, AND EITHER WIN THEIR PLAUE OR DIE OF NEGLEUTK
End decryption? n
Enter first letter to swap: C
Enter second letter to swap: U
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FIXED AND INVARIABLE, NOR DOES IT
DEPEND ON ANY OTHER AUTHORITY THAN GENERAL AGREEMENTK AT THE PRESENT DAY THERE IS
PRACTICALLY UNANIMOUS AGREEMENT AS TO THE SPELLING OF MOST WORDSK IN THE LIST BELOW, FOR
EXAMPLE, .RIME. FOR .RHYME. IS THE ONLY ALLOWABLE VARIATION" ALL THE OTHER FORMS ARE
CO-EXTENSIVE WITH THE ENGLISH LANGUAGEK AT ANY GIVEN MOMENT, HOWEVER, A RELATIVELY
SMALL NUMBER OF WORDS MAY BE SPELLED IN MORE THAN ONE WAYK GRADUALLY, AS A RULE, ONE OF
THESE FORMS COMES TO BE GENERALLY PREFERRED, AND THE LESS CUSTOMARY FORM COMES TO LOO'
OBSOLETE AND IS DISCARDEDK FROM TIME TO TIME NEW FORMS, MOSTLY SIMPLIFICATIONS, ARE
INTRODUCED BY INNOVATORS, AND EITHER WIN THEIR PLACE OR DIE OF NEGLECTK
End decryption? n
Enter first letter to swap: K
Enter second letter to swap: .
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FIXED AND INVARIABLE, NOR DOES IT
DEPEND ON ANY OTHER AUTHORITY THAN GENERAL AGREEMENT. AT THE PRESENT DAY THERE IS
PRACTICALLY UNANIMOUS AGREEMENT AS TO THE SPELLING OF MOST WORDS. IN THE LIST BELOW, FOR
EXAMPLE, KRIMEK FOR KRHYMEK IS THE ONLY ALLOWABLE VARIATION" ALL THE OTHER FORMS ARE
CO-EXTENSIVE WITH THE ENGLISH LANGUAGE. AT ANY GIVEN MOMENT, HOWEVER, A RELATIVELY
SMALL NUMBER OF WORDS MAY BE SPELLED IN MORE THAN ONE WAY. GRADUALLY, AS A RULE, ONE OF
THESE FORMS COMES TO BE GENERALLY PREFERRED, AND THE LESS CUSTOMARY FORM COMES TO LOO'
OBSOLETE AND IS DISCARDED. FROM TIME TO TIME NEW FORMS, MOSTLY SIMPLIFICATIONS, ARE
INTRODUCED BY INNOVATORS, AND EITHER WIN THEIR PLACE OR DIE OF NEGLECT.
End decryption? n
Enter first letter to swap: K
Enter second letter to swap: '
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FIXED AND INVARIABLE, NOR DOES IT
DEPEND ON ANY OTHER AUTHORITY THAN GENERAL AGREEMENT. AT THE PRESENT DAY THERE IS
PRACTICALLY UNANIMOUS AGREEMENT AS TO THE SPELLING OF MOST WORDS. IN THE LIST BELOW, FOR
EXAMPLE, 'RIME' FOR 'RHYME' IS THE ONLY ALLOWABLE VARIATION" ALL THE OTHER FORMS ARE
CO-EXTENSIVE WITH THE ENGLISH LANGUAGE. AT ANY GIVEN MOMENT, HOWEVER, A RELATIVELY
SMALL NUMBER OF WORDS MAY BE SPELLED IN MORE THAN ONE WAY. GRADUALLY, AS A RULE, ONE OF
THESE FORMS COMES TO BE GENERALLY PREFERRED, AND THE LESS CUSTOMARY FORM COMES TO LOOK
OBSOLETE AND IS DISCARDED. FROM TIME TO TIME NEW FORMS, MOSTLY SIMPLIFICATIONS, ARE
INTRODUCED BY INNOVATORS, AND EITHER WIN THEIR PLACE OR DIE OF NEGLECT.
End decryption? n
Enter first letter to swap: "
Enter second letter to swap: ;
Decrypted string: THE SPELLING OF ENGLISH WORDS IS NOT FIXED AND INVARIABLE, NOR DOES IT
```

```
DEPEND ON ANY OTHER AUTHORITY THAN GENERAL AGREEMENT. AT THE PRESENT DAY THERE IS
PRACTICALLY UNANIMOUS AGREEMENT AS TO THE SPELLING OF MOST WORDS. IN THE LIST BELOW, FOR
EXAMPLE, 'RIME' FOR 'RHYME' IS THE ONLY ALLOWABLE VARIATION; ALL THE OTHER FORMS ARE
CO-EXTENSIVE WITH THE ENGLISH LANGUAGE. AT ANY GIVEN MOMENT, HOWEVER, A RELATIVELY
SMALL NUMBER OF WORDS MAY BE SPELLED IN MORE THAN ONE WAY. GRADUALLY, AS A RULE, ONE OF
THESE FORMS COMES TO BE GENERALLY PREFERRED, AND THE LESS CUSTOMARY FORM COMES TO LOOK
OBSOLETE AND IS DISCARDED. FROM TIME TO TIME NEW FORMS, MOSTLY SIMPLIFICATIONS, ARE
INTRODUCED BY INNOVATORS, AND EITHER WIN THEIR PLACE OR DIE OF NEGLECT.
End decryption? yes
```

What To Submit

You must submit all your files on codePost (https://codepost.io/). The file you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`game_of_life.py`
`my_game_of_life.gif`
`coin_utils.py`
`coin_crypt.py`
`README.txt` In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.
Remember that this assignment like all others is an `individual` assignment and must represent the entirety of your own work. You are permitted to verbally discuss it with your peers, as long as no written notes are taken. If you do discuss it with anyone, please make note of those people in this `README.txt` file. If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.