COMP 202

Winter 2022

Assignment 2

Due: Sunday, February 20^{th}, 11:59 p.m.

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

Question 1:	60 points
Question 2:	40 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, allowing the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while grading, then that increases the chance of them giving out partial marks. :)

To get full marks, you must follow all directions below:

- Make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, a 50% penalty per question will be applied.
- Make sure that your code **runs without errors**. Code with errors will receive a very low mark.
- Write your name and student ID in 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 code.** 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.
- **Up to 30% can be removed for bad indentation of your code, omission of comments, and/or poor coding style (as discussed in class).**

Hints & tips

- **Start early.** Programming projects always take more time than you estimate!
- Do not wait until the last minute to submit your code. **Submit early and often**—a good rule of thumb is to submit every time you finish writing and testing a function.
- Write your code **incrementally**. Don't try to write everything at once. That never works well. Start off with something small and make sure that it works, then add to it gradually, making sure that it works every step of the way.
- Read these instructions and make sure you understand them thoroughly before you start. Ask questions if anything is unclear!

- Seek help when you get stuck! Check our discussion board first to see if your question has already been asked and answered. Ask your question on the discussion board if it hasn't been asked already. Talk to your TA during office hours if you are having difficulties with programming. Go to an instructor's office hours if you need extra help with understanding a part of the course content.
  - At the same time, beware not to post anything on the discussion board that might give away any part of your solution—this would constitute plagiarism, and the consequences would be unpleasant for everyone involved. If you cannot think of a way to ask your question without giving away part of your solution, then please drop by our office hours.
- If you come to see us in office hours, please do not ask "Here is my program. What's wrong with it?" We expect you to at least make an effort to start to debug your own code, a skill which you are meant to learn as part of this course. And as you will discover for yourself, reading through someone else's code is a difficult process—we just don't have the time to read through and understand even a fraction of everyone's code in detail.
  - However, if you show us the work that you've done to narrow down the problem to a specific section of the code, why you think it doesn't work, and what you've tried to fix it, it will be much easier to provide you with the specific help you require and we will be happy to do so.

Revisions

Feb 8: Corrected typos and added clarifications in Question 2, and updated the provided `wordle_utils.py` file.

Feb 10: Question 1: No need to use nested loops for `flip_vertical` function.

Feb 14: Question 2: Added `random.seed` in the examples for functions `generate_random_wordle` and `choose_mode_and_wordle`. Also added examples for the functions `main` and `play` and a note for `generate_random_wordle`.

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 `swap` which takes as input two int values `x` and `y`. Your function should do 3 things:

1. Print the value of `x` and `y`
2. Swap the values of the variables `x` and `y`, so that whatever was in `x` is now in `y` and whatever was in `y` is now in `x`
3. Print the value of `x` and `y` again.

Here is an example of what your function should print when called (in the shell):

```
>>> swap(3, 4)
inside swap: x is:3 y is:4
inside swap: x is:4 y is:3
```

Warm-up Question 2 (0 points)

Consider the program you have just written. Create two global integer variables in the main body of your program. Call them `x` and `y`. Assign values to them and call the swap function you wrote in the previous part using `x` and `y` as input parameters.

After calling the `swap()` function —inside the main body— print the values of `x` and `y`. Are they different than before? Why or why not?

Warm-up Question 3 (0 points)

Create a function called `counting` that takes as input a positive integer and counts up to that number. For example:

```
>>> counting(10)
Counting up to 10: 1 2 3 4 5 6 7 8 9 10
```

Warm-up Question 4 (0 points)

Modify the last function by adding an additional input that represents the step size by which the function should be counting. For example:

```
>>> counting(25, 3)
Counting up to 25 with a step size of 3: 1 4 7 10 13 16 19 22 25
```

Warm-up Question 5 (0 points)

Write a function `replace_all` which takes as input a string and two characters. If the second and third input string do not contain exactly one character the function should raise a `ValueError`. Otherwise, the function returns the string composed by the same characters of the given string where all occurrences of the first given character are replaced by the second given character. For example, `replace_all("squirrel", "r" , "s")` returns the string `"squissel"`, while `replace_all("squirrel", "t", "a")` returns the string `"squirrel"`. Do not use the method `replace` to do this.

Warm-up Question 6 (0 points)

Write a module with the following global variables:

```
lower_alpha = "abcdefghijklmnopqrstuvwxyz"
upper_alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

In this module write a function `make_lower` which takes a string as input and returns a string containing the same characters as the input string, but all in lower case. For example, `make_lower("AppLE")` returns the string `"apple"`. Do not use the method `lower` to do this. Hint: note that characters from the English alphabet appear in the same position in the two global variables.

Warm-up Question 7 (0 points)

Write a function `split_at_space` that takes as input a string of words with spaces in between each word and returns a list of the words in the string.

```
>>> split_at_space('aardvark butterfly carrot')
['aardvark', 'butterfly', 'carrot']
```

Warm-up Question 8 (0 points)

Write a function `generate_random_list` which takes an input an integer `n` and returns a list containing `n` random integers between 0 and 100 (both included). Use the `random` module to do this.

Warm-up Question 9 (0 points)

Write a function `sum_numbers` which takes as input a list of integers and returns the sum of the numbers in the list. Do not use the built-in function `sum` to do this.

Part 2

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

The main learning objectives for this assignment are:

- Correctly define and use simple functions.
- Solidify your understanding of the difference between `return` and `print`.
- Generate and use random numbers inside a program.
- Understand how to test functions that contain randomness.
- Correctly use loops and understand how to choose between a while and a for loop.
- Solidify your understanding of how to work with strings: how to check for membership, how to access characters, how to build a string with accumulator patterns.
- Begin to use very simple lists.
- Learn how to use functions you have created in a different module.

Note that this assignment is designed for you to be practicing what you have learned in the videos up to and including Lecture 20 (Lists 2). For this reason, you are NOT allowed to use anything seen after Lecture 20 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 three (3) examples of calls to the function. 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 test 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 will use inputs not seen in the examples. You should make sure that your functions work for all the different possible scenarios. Also, when testing your code, know that mindlessly plugging in various different inputs is not enough—it's not the quantity of tests that matters, it's having tests that cover all of the possible scenarios, and that requires thinking about possible scenarios.

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 contains function calls in the main body **will automatically fail the tests on codePost and thus 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. You can also test your functions by calling them from the shell. Please review what you have learned in Lecture 14 (Modules) if you'd like to add code to your modules which executes only when you run your files.

Safe Assumptions

For all questions on this assignment, you can safely assume that the **type** of the inputs (both to the functions and those provided to the program by the user) will always be correct. For example, if a function takes as input a string, you can assume that a string will always be provided. At times you will be required to do some input validation, but this requirement will always be clearly stated. Otherwise, your functions should work with any possible input that respect the function's description. For example, if the description says that the function takes as input a positive integer, then it should work with all integers greater than 0. If it mentions an integer, then it should work for *any* integer. Make sure to test your functions for edge cases!

Code Repetition

One of the main principles of software development is DRY: Don't Repeat Yourself. One of the main ways we can avoid repeating ourselves in code is by writing functions, then calling the functions when necessary, instead of repeating the code contained within them. Please pay careful attention in the questions of this assignment to not repeat yourself, and instead call previously-defined functions whenever appropriate. As always, you can also add your own helper functions if need be, with the intention of reducing code repetition as much as possible.

Question 1: The Secret Chatroom (60 points)

In the past decade, instant messaging services have become a constant and subtly insidious part of our lives. Where before one may have spoken over the phone or perhaps sent a plain old SMS text message to talk with friends, this activity now requires accounts on a wide variety of platforms, such as WhatsApp, iMessage, Facebook Messenger, Discord, Zoom and others. Even some forward-looking university classes have set up their own chat platforms through Slack or (shudder) Microsoft Teams. But in COMP 202 we are taking things to the next level, for we will have our own chatroom free from the confines of the *mode du jour*. But only those whose code is in working order may enter...

In this question, we will write a series of functions that deal with manipulation of black and white images, so that we may gain access to a secret chatroom which accepts users only when their code responds successfully to basic image manipulation requests.

A digital image can be thought of as a two-dimensional grid of '**pixels**' – little squares, each with their own color, arranged in rows and columns. You can see the individual pixels of an image for yourself by opening an image file on your computer and zooming in to the maximum level.

When writing a program that deals with images, we typically use a data type that can deal with two-dimensional data, like a nested list (list of lists), but we will only see these things later in the course. Therefore, for this assignment question, we will represent an image by a string. The string will contain one digit for every pixel in the image: either a 0, for a black pixel, or a 1, for a white pixel.

Although an image is two-dimensional, our image string will be a flat (1-dimensional) sequence of 0's and 1's. All the pixels of the first row of the image will appear first in the string, followed by all the pixels of the second row, third row, and so on. For instance, here is an example image string, which contains nine pixels (eight black and one white):

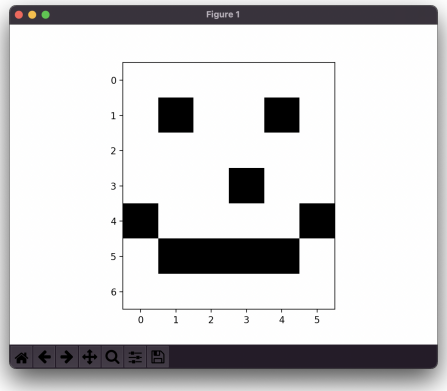
```
'000010000'
```

Along with the image string, we will sometimes give you the height (number of rows) and width (number of columns) of an image. We might say that the height and width of the above image string are three, meaning a 3x3 image (although in general, image strings can be rectangular and not always square). Then the first and last row would be black, and the middle row would be black except for one white square in the middle (row 1, column 1, if we start counting at 0).

We will be dealing with image strings a lot in this question, so we provide to you with this PDF a file called `image_utils.py` which contains a function `show_image` that lets you display an image string to the screen as a black and white image. The function takes three parameters: the image string and the integer height and width of the image.

Here is an example of a call to the function and what it displays:

```
>>> show_image('111111101101111111111011011110100001111111', 7, 6)
```



Since you will be dealing with images a lot, it's strongly suggested for you to make your own image strings with different properties in order to thoroughly test your functions. Of course, creating image strings could be considered as a creative act, so, as in Assignment 1, we are once again asking for your artistic contribution: if you would like, you can optionally submit (not for credit) with your assignment an image string in a file called `image.txt`. We may showcase certain of the images in our Slack later in the term.

Before writing any code, we must install a number of Python libraries that will help to handle the displaying of images and network connection abilities. A library is a Python module (or a collection of modules) created by a third-party developer (i.e., someone other than the Python development team). We will learn a bit more about libraries later this term. You will not be using them directly for this assignment; do not import them yourself and do not use any functions from them for this assignment. They are only used by the code files that we have provided to you with this PDF.

To install a library, please follow these steps:

1. In Thonny, go to Tools > Manage packages... in the menu bar.
2. Enter the module name in the text field and press the Search on PyPI button.
3. Click on the appropriate search result.
4. Click Install. The library will download and install. Depending on its size, it may take a few seconds.

Here are the libraries you must install:

- `numpy`
- `Pillow`
- `matplotlib`
- `websocket-client` (Note: **not** websocket-client**2**)

We can now start implementing our functions. First we will write the functions that deal with image manipulation. Write all functions for this question in a file `image_proc.py`.

Note: For all functions in this question, you may **not** use nested lists nor **any** string or list methods. If you do, a 50% penalty will be applied to your total point value for this question. (You should also not need to use lists at all for this question.)

- **`invert`**: takes an image string as input, and returns the inverted image string. Inverting an image means to switch each pixel to the opposite color. In the case of a black and white image, black (0) inverted becomes white (1), and white inverted becomes black.

```
>>> invert('0111')
'1000'
```

- **`flip_horizontal`**: takes an image string and integer height and width of the image as input, and returns the image string flipped horizontally (i.e., where each row of pixels is reversed).

```
>>> flip_horizontal('011100', 2, 3)
'110001'
```

- **`flip_vertical`**: takes an image string and integer height and width of the image as input, and returns the image string flipped vertically (i.e., where each column of pixels is reversed).

```
>>> flip_vertical('011100', 2, 3)
'100011'
```

- **crop**: takes an image string, integer height and width of the image, top left x and y coordinates (integers) and integer height and width of the crop rectangle as input, and returns a cropped rectangle (as string) from the image that begins at the given x and y coordinates (starting from the top left) and is of the given height and width.

  Hint: You may have to use a nested loop (loop within a loop) here. Try doing this problem on paper first to find the steps needed, and then implement it into code. A diagram could help you to find the indices of the crop rectangle.

```
>>> crop('000111000', 3, 3, 1, 1, 1, 1)
'1'

>>> crop('000111000', 3, 3, 1, 1, 2, 2)
'1100'
```

The next set of functions that you will write (in the same file) will deal with image compression. Our chat server will be sending compressed images to your program, so you must be able to decompress them in order to manipulate them.

Compression is the process of taking a file and performing some operation on it to reduce its size without losing any data. For instance, creating `zip` file stores multiple files into one with a possibly lower file size. Compression is important especially when transferring files over the net, since a smaller file size will result in a faster transfer.

There are many compression algorithms, some much more sophisticated and advanced than others. We will use a simple compression algorithm here that is specially made for our image strings. First, we will write a helper function, and then follow it up with a function to compress and to decompress.

- **find_end_of_repeated_substring**: takes a string `s`, an integer index `i` into the string, and a single character `c` as input. You can assume that `s[i] = c`. Looks through the string starting at the given index, and returns the index (as an integer) of the last consecutive occurrence of the character `c`. That is, as soon as you come across a character not equal to `c` (after starting at `i`), then return the previous index.

```
>>> find_end_of_repeated_substring('aaaaabbbbaababaccc', 0, 'a')
4

>>> find_end_of_repeated_substring('011', 1, '1')
2
```

- **compress**: takes an image string as input and returns the compressed image string. If the image string contains any characters other than 0 or 1, then return the empty string.

  Compression is performed as follows:

  1. An empty string, to store the compressed image is created. Then, the first character of the image string is appended to it followed by a space.
  2. The image string is analyzed to determine the number of consecutive same characters from the beginning. This count is then added to the new (compressed) string, followed by a space.
  3. The step above repeats for each successive consecutive sequence of 0's and 1's until all numbers have been counted. At that point, remove the trailing space and return the compressed string.

  For instance, if the image string to be compressed is `'0000110'`, then we first put a 0 (since the first character of the image string is a 0); followed by a 4 (since there are four consecutive zeroes); followed by a 2 (since there are two consecutive ones); followed by a 1 (since there is one consecutive zero), with spaces in between each. Thus the compressed image string in this case would be `'0 4 2 1'`.

Hint: Think about how you could use the above helper function in this function.

```
>>> compress('00011')
'0 3 2'

>>> compress('0001011111111')
'0 3 1 1 8'

>>> compress('1')
'1 1'
```

- `decompress`: takes a compressed image string as input and returns the decompressed image string. If the compressed image string contains any characters other than a digit from 0-9 or a space, then return the empty string.

  Decompression is performed by undoing the steps of the compression listed above. Note that this function can be a little challenging to write. You may want to sketch out some plans on paper first. (Nested loops are not needed.)

```
>>> decompress('0 3 2')
'00011'

>>> decompress('0 3 1 1 8')
'0001011111111'

>>> decompress('1 1')
'1'
```

Finally, we will write a function that takes a series of image manipulation commands, and performs each of them in turn on a given image string. This function will be called by our chat server to check if your functions are working properly.

- `process_command`: takes a command string, image string, and integer height and width of the image as input, and returns the resulting image string after the commands have been performed.

  A command string is a string of space-separated command words. Each command word corresponds to an image manipulation function: INV for invert, FH for flip horizontal, FV for flip vertical, CR for crop, DC for decompress and CP for compress.

  The crop (CR) command is special: it will be followed by four integers separated by commas, corresponding to the four extra arguments that must be given to the `crop` function. E.g., a crop command may look like `'CR10,12,2,3'`, where 10 and 12 are the top-left x and y coordinate and 2 and 3 are the height and width of the crop rectangle.

  Each command in turn should be performed on the resulting image string of the previous command. That is, if the command string is `'DC INV CP'`, then the image string that was passed as input to the function should first be decompressed; then, the decompressed image should be inverted; and finally, the inverted image should be compressed, and the resulting compressed image should be returned.

  Note that there could be any number of commands in the command string, and the same command could be repeated at different points in the string.

  You can assume that the command string will be valid (i.e., that it contains valid commands separated by spaces, and that a decompress command will only occur either as the first command or after a compress command, and that a compress command will only occur either as the last command or after a decompress command).

Hint: The hard part of this function is to separate the commands from the command string. The crop command will also require more work, since you must get the four integers that follow it. (There you might consider whether the same pattern used to extract the commands may be used also to extract the integers.)

```
>>> process_command('INV', '101001', 3, 2)
'010110'

>>> process_command('INV FH', '101001', 3, 2)
'101001'
```

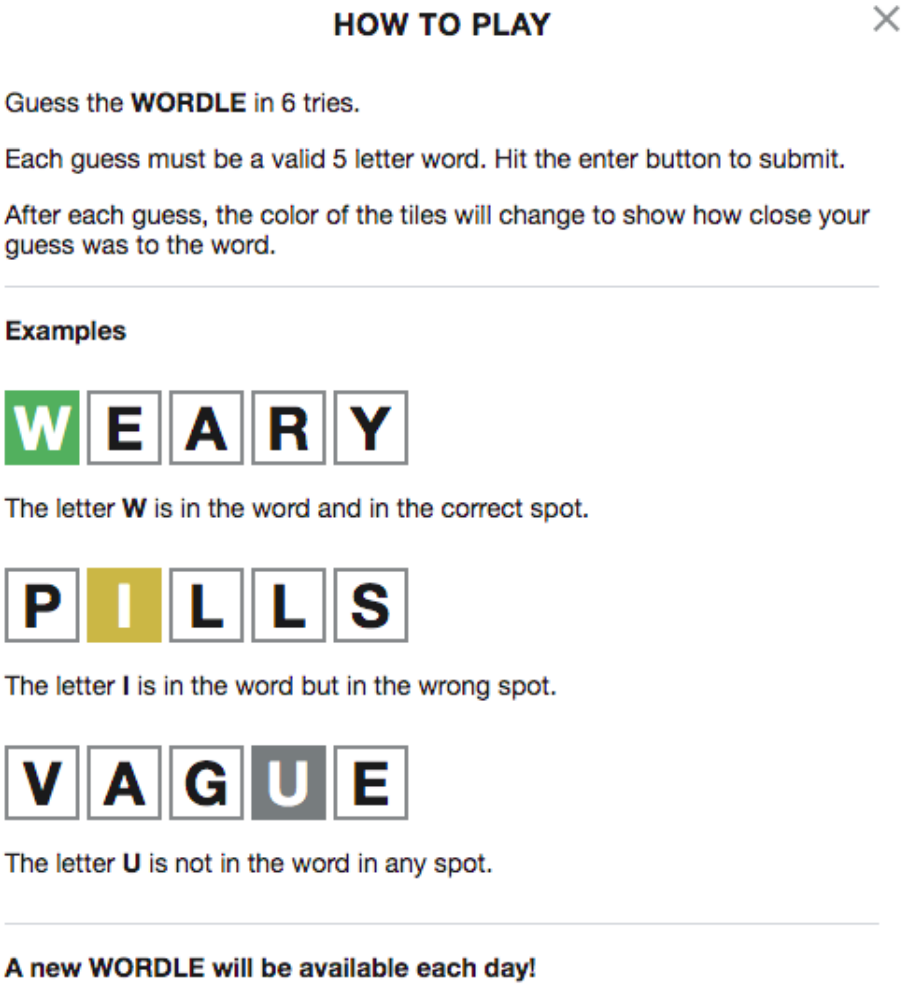
After implementing these functions, you can then try to connect to our chat server. To do so, open the file `chat.py` that is provided to you with this PDF, and change the `HANDLE` variable to your username, then run the file.

A chat window will appear with a text field where you can send messages. But beware: approximately every 30 seconds, the server will send each connected user an image string and command string, and will automatically call your `process_command` function on them and receive the result returned by the function. If the function does not return the correct result, then your connection to the chat server will be automatically terminated.

Finally, a note to please be respectful to others when participating in the chat. We will be moderating the chat.

Question 2: Wordle (40 points)

Have you heard about the recent viral game Wordle? If yes, then chances are you don't go a day without playing it. If not, then go check it out here. Wordle is a simple word game where you have to guess the five letter word which is the word of the day. Here are the rules of the game from its creators.



There is a new "wordle" every day – you can't play it multiple times a day. Moreover, you are always playing against the game, not with a friend. To solve these issues we are going to make our own version of this game so that we can play as many times as we want and we are going to make it so that we can play it with a friend. (Yes, we've already learned enough tools to be able to write a game!)

We provide you with the database of words used by Wordle itself (`wordle-dictionary.txt`). Since we haven't learned yet how to work with files, we will give you a Python module which reads the database and stores the words in a list (`wordle_utils.py`). You need to import this module in your file and call the function `load_words` to get all the words from the database. The function takes no parameters and returns a list of strings.

Remark. Make sure to save the `wordle_utils.py` and `wordle-dictionary.txt` in the same folder as your solution file, which must be called `wordle.py`.

We are going to play the game from the shell. In our version of the game, we are going to have two modes of playing – with 1 player, or 2 players.

- **1 player mode**: in this mode the code selects a random five letter word from the list of words and the player must guess the word.

- **2 player mode**: in this mode Player 1 inputs a five letter word into the shell, then, after 2 seconds Player 1's input disappears from the shell (we will explain how to do this below) and Player 2 must guess the word entered by Player 1. If Player 1 enters a word which doesn't have 5 letters or cannot be found in the given list of words, then the game must ask the player to enter another word which is valid.

Implementation. As stated before, write all your code for this question in a file called `wordle.py`.

First, define the following global variables at the top of your file:

- `WORDLE_WORD_LENGTH = 5` (the length of the words in the game)
- `MAX_NUM_OF_GUESSES = 6` (how many incorrect guesses the player is allowed to enter)
- `CHAR_GREEN = '\x1b[6;30;42m'`
- `CHAR_YELLOW = '\x1b[6;30;43m'`
- `CHAR_GRAY = '\x1b[6;30;47m'`
- `CHAR_END = '\x1b[0m'`

The last four variables are there to help you to print colored letters in the shell. These variables are the ANSI codes for respective colors. To print colored string in the shell we need to "sandwich" the string by placing it in between the ANSI color code and the `CHAR_END` variable (which indicates the end of the coloring instruction). Here are some examples for you to try in the shell:

```
>>> print(CHAR_YELLOW + 'apple' + CHAR_END)
apple
>>> print(CHAR_GREEN + 'apple' + CHAR_END + ' ' + CHAR_GRAY + 'pie' + CHAR_END)
apple pie
```

Before going into the details of implementing the game, we ask you to implement the following helper functions first.

- `is_valid_word`: takes two inputs – a word as string and list of string as a list of words. Returns `True` if the word is a five letter word and belongs to the given list of words, otherwise the function returns `False`.

  ```
  >>> is_valid_word('about', ['abounds', 'about', 'abouts', 'above', 'aboveboard'])
  True
  >>> is_valid_word('about', ['abounds', 'abouts', 'above', 'aboveboard'])
  False
  ```

- `print_string_list`: takes a list of strings and prints each string in the list on a new line.

  ```
  >>> print_string_list(['abounds', 'about', 'abouts', 'aboveboard', 'abovedeck'])
  abounds
  about
  abouts
  aboveboard
  abovedeck
  ```

- **`color_string`**: Takes two strings: a word to be colored, and a color. The function should return the "colored" string, meaning, it returns a new string where the word to be colored is concatenated in between the corresponding ANSI code of the color and `CHAR_END`. The color string can be either `'green'`, `'yellow'` or `'gray'`. If the color string is none of the above, the function prints `'Invalid color.'` and returns the original word string (the first parameter).

```
# The function returns the string with the word in between the coloring instructions.
>>> color_string('about', 'green')
'\x1b[6;30;42mabout\x1b[0m'

# If we use a print statement on the returned string, we will see it in color.
>>> s = color_string('about', 'green')
>>> print(s)
about

>>> color_string('about', 'black')
Invalid color.  # Note: This line is printed due to a print statement in the function,
'about'         # while this line indicates the return value of the function.
```

- **`get_all_5_letter_words`**: Takes a list of strings representing a list of words and returns a new list consists of only `WORDLE_WORD_LENGTH` length words.

```
>>> get_all_5_letter_words(['abs', 'about', 'abouts', 'above', 'aboveboard', 'aloft'])
['about', 'above', 'aloft']

>>> get_all_5_letter_words(['abounds', 'about', 'abouts', 'aboveboard', 'abovedeck'])
['about']
```

Next, we ask you to implement the following functions to implement the logic of the game.

Note: Some of these functions ask the player to input a word. While this input words can have both be capital case and lower case letters, all the words in the dictionary are stored in lower case. Thus, although entering words with capital case letters is allowed, the game must be played in lower case, meaning all the words which will be printed in the shell as guess words should be in lower case. See the examples for the functions `play_with_word` and `input_wordle`.

- `main`: takes no parameters and returns nothing. The function first loads the list of all words by calling the function `load_words` from the `wordle_utils` module. Then, from that list it filters and keeps only 5 letter words using helper functions. Then finally it calls the function `play` passing as parameter the list of 5 letter words. As a result, the `main` function prints exactly what `play` prints when calling it.

  Note: there is no need to include examples in the docstring for this function. But the type contract and description are still needed.

```
>>> random.seed(100)
>>> main()
 #The following are the messages printed in the shell for this function call.

Enter the number of players: 1
Enter a guess: arose
a r ose
Enter a guess: rapid
a r ose
r ap id
Enter a guess: ridden
Not a valid word, please enter a new one.
Enter a guess: drain
a r ose
r ap id
dr a in
Enter a guess: drink
a r ose
r ap id
dr a in
drink
You won! It took you 4 guesses.
```

- **play**: takes a list of strings as input which represents the list of all words. The function must choose the play mode and get the wordle – word of the game. Upon getting the wordle, it should call the function play_with_word with respective parameters, and finally it should the print the final message to indicate whether the user won or lost using the function print_final_message. As a result, the function play prints exactly what the three functions print when calling them.

  Note: there is no need to include examples in the docstring for this function. But the type contract and description are still needed.

```
>>> play(['cable', 'cater', 'crane','carve', 'caper', 'calls'])
 #The following are the messages printed in the shell for this function call.

    Enter the number of players: 2

    ***** Player 1's turn. ***** # Player 1 enters 'caper'



    ***** Player 2's turn. *****

    Enter a guess: Crane
    c ra n e
    Enter a guess: cArvE
    c ra n e
    ca r v e
    Enter a guess: cater
    c ra n e
    ca r v e
    ca t er
    Enter a guess: caper
    c ra n e
    ca r v e
    ca t er
    caper
    You won! It took you 4 guesses.
```

- `choose_mode_and_wordle`: takes a list of strings as input which represents the list of all words and returns the wordle – the word which is going to be solution of the game.

  The function first needs to ask the user which mode of playing they would like. Namely, it should ask the user to input the mode with the prompt message `"Enter the number of players: "`. The user should enter 1 for '1 player mode' and 2 for '2 player mode'. If the user's input is not 1 or 2, then the function should print `"Wordle can be played with 1 or 2 players. Please only enter 1 or 2."` and should ask the user to input again.

  If the user chooses the 1 player mode, then the function should generate the wordle randomly from the given list of words.

  If the user chooses the 2 player mode, then the function should print `"\n***** Player 1's turn. ***** \n"` and ask the Player 1 to input the wordle, then print `"\n***** Player 2's turn. ***** \n"`.

```
>>> random.seed(20)
>>> choose_mode_and_wordle(['about', 'above', 'aloft', 'aeons'])
Enter the number of players: 1
'above'  # the return value of the function call

>>> choose_mode_and_wordle(['about', 'above', 'aloft', 'aeons'])
Enter the number of players: 2
***** Player 1's turn. *****

***** Player 2's turn. *****

'aeons'  # the return value of the function call
```

- `input_wordle`: takes a list of strings as input which represents the list of all words and returns the wordle. Asks the user to input a word with the message `"Input today's word: "`, then it should get erased from the shell immediately. That's why you can't use the usual function `input` here. We are providing you with a special function `input_and_hide` in `wordle_utils.py` which will hide the input after 2 seconds – use that function instead.

  The function should check whether the input is a valid word. If it's not a valid word, the function should print `'Not a valid word, please enter a new one.'`, and should continue to ask the user until it's a valid word.

  Note: there is no need to include examples in the docstring for this function. But the type contract and description are still needed.

```
>>> input_wordle(['about', 'above', 'aloft', 'aeons'])
        Input today's word: ABOUT
'about'  # the return value of the function call

>>> input_wordle(['about', 'above', 'aloft', 'aeons'])
        Input today's word: mount
Not a valid word, please enter a new one.
        Input today's word: aloft
'aloft'  # the return value of the function call
```

- `generate_random_wordle`: takes a list of strings as input which represents the list of all words. The function picks a random string from the list and returns it.

  Note: You should use the function `random.randint`.

```
>>> random.seed(100)
>>> generate_random_wordle(['about', 'above', 'aloft', 'aeons'])
'above'

>>> random.seed(1000)
>>> generate_random_wordle(['about', 'above', 'aloft', 'aeons'])
'aeons'
```

- `play_with_word`: takes the solution word as string and a list of words as a list of strings. The function asks the player to input a guess word with a prompt message `'Enter a guess:'`, then the guess must be colored respective to the solution word. If it's not a valid word, the function should print `'Not a valid word, please enter a new one.'`, and should continue to ask the user until it's a valid word.

  After each guess the function prints the colored guesses made so far using the helper functions. The function keeps doing this for at most `MAX_NUM_OF_GUESSES` times; it interrupts early if the player guesses the solution. The function returns the number of words the player entered to guess the solution. If the player didn't guess the solution word in `MAX_NUM_OF_GUESSES` tries, then it returns `MAX_NUM_OF_GUESSES + 1`.

```
>>> play_with_word('caper', ['cable', 'cater', 'crane', 'carve', 'caper', 'calls'])
 #The following are the messages printed in the shell for this function call.
Enter a guess: Crane
c ra n e
Enter a guess: cArvE
c ra n e
ca r v e
Enter a guess: cater
c ra n e
ca r v e
ca t er
Enter a guess: caper
c ra n e
ca r v e
ca t er
caper

4 #The integer returned by the function.
```

- `compare_and_color_word`: takes two strings, the first one is the guessed word and the second one the solution word. The function must compare the guessed word to the solution word letter by letter and color each letter accordingly. Namely, if a letter from the guessed word doesn't appear in the solution, then color it gray. If the letter appears in the solution word in the same position then color it green, otherwise if it appears but not in the same position color it yellow. To color the letters use the helper functions. The function returns the colored string.

```
>>> compare_and_color_word('mount', 'about')
'\x1b[6;30;47mm\x1b[0m\x1b[6;30;43mo\x1b[0m\x1b[6;30;43mu\x1b[0m\x1b[6;30;47mn\x1b
[0m\x1b[6;30;42mt\x1b[0m' #The string returned by the function.

>>> compare_and_color_word('plate', 'petal')
'\x1b[6;30;42mp\x1b[0m\x1b[6;30;43ml\x1b[0m\x1b[6;30;43ma\x1b[0m\x1b[6;30;43mt\x1b[0m
\x1b[6;30;43me\x1b[0m' #The string returned by the function.
```

- `print_final_message`: takes two inputs:
  - an int – the number of words it took the user to guess the word
  - a string – the wordle (the solution word)

  The function returns nothing. It prints the final message of the game indicating whether the player has lost or won. The function decides whether the player has won or lost according to the guessed words. If the player lost the function should print `"You lost!"` and it should print the correct word colored in green.

  Otherwise, the player won, so the function should print `"You won!"`. In this case the function should print the number of words it took the player to win, i.e. `"It took you 3 guesses."` or `"It took you 1 guess."`

```
>>> print_final_message(3, 'about')
You won! It took you 3 guesses.

>>> print_final_message(7, 'about')
You lost! The word was about
```

What To Submit

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

`image_proc.py`
`image.txt` In this file, you can include an image string that you created if you like. If you do not wish to do so, then please submit an empty file.
`wordle.py`
`README.txt` In this file, you can tell the TA about any issues you ran into while 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.

You may make as many submissions as you like, but we will only grade your final submission (all prior ones are automatically deleted).

Note: If you are having trouble, make sure the names of your files are exactly as written above.

Assignment debriefing

In the week following the due date for this assignment, you will be asked to meet with a TA for a 10-15 minute meeting. In this meeting, the TA will grade your submission and discuss with you what you should improve for future assignments.

Only your code will determine your grade. You will not be able to provide any clarifications or extra information in order to improve your grade. However, you will have the opportunity to ask for clarifications regarding your grade.

You may also be asked during the meeting to explain portions of your code. Answers to these questions will again not be used to determine your grade, but inability to explain your code may be used as evidence to support a charge of plagiarism later in the term.

Details on how to schedule a meeting with the TA will be shared with you close to the due date of the assignment.

If you do not attend the meeting, you will not receive a grade for your assignment.