Due Mar 9 by 11:59pm **Points** 100 **Available** after Feb 20 at 9:45pm

Home

Resources

Homework 5: Strings, Lists & Everything up to now

Time to put everything together! This week's homework will give you the opportunity to practice "connecting the dots" and building a larger solution with just about everything we've covered in the course thus far. It's a fun "mini project" that will help solidify many of the concepts we've been practicing in previous homework assignments.

Written Component (20% of your homework score)

- Filename: written.txt

Written #1

For each of the Python snippets below, what will be printed to the terminal? Be specific about linebreaks and indentation where applicable.

1A

```
1  s = "coco"
2  t = "puffs"
3  print(2 * (s + t))
```

1B

```
1  s = "coco"
2  t = "puffs"
3  print(s + (2 * t))
```

1C

```
1  s = "coco"
2  t = "puffs"
3  print("My Dog" in 2 * (s + t))
```

Written #2

Write **one line o**f Python code that will return, from the string *s*, every third letter working backwards from the end. Here are some examples -- the same one line of code should work on all of them:

s = "madamimadam"

 # should return "mama"

s = "helloworld"

 # should return "dolh"

s = "racecar"

 # should return "rer"

Written #3

For the string **s = "boston red sox"**, what does each of the Python expressions produce? If it would produce an error, specify what type of error.

3A **s[11:13]**

3B **s[11:14]**

3C **s[11:15]**

Programming Component (80% of this HW)

Kudos on reaching this milestone in the course! You've got all the tools and knowledge you need to create some very interesting and robust solutions. There is only ONE programming assignment this week. We want you to focus on computational thinking, procedural decomposition, good function design and good code structure. Go ahead and use any loops you like, and any list/string functions we've covered in class, or that you've found yourself. Some that we found helpful for this HW:

- *in:* Tells you whether a list/string contains a value
- *split:* Turns a string into a list
- *join*: Turns a list into a string
- *index:* Returns the position in a list of the first instance of a given value

Program: ReMix Master

Files:

- `remix_master.py` (and any other "helper files" you develop)
- Starter file: **`music.py`** ⬇ **(https://northeastern.instructure.com/courses/102990/files/13111441/download?download_frd=1)** (data file for my songs and playlist)

An earlier cohort of ALIGNers included a recording engineer. I'm not musically inclined, but I thought it was pretty cool to have a person from the music industry in the program.

For this assignment, we're going to build our own song remixing solution.

As an example: given the following American children's song (this is ONE of the 3 songs I'm giving you in the starter code that includes the data you'll use for your program):

```
SONG = ['old macdonald had a farm - ee-i-ee-i-o.',

        'and on that farm he had a cow - ee-i-ee-i-o.',

        'with a moo moo here and a moo moo there',

        'here a moo - there a moo - everywhere a moo moo',

        'old macdonald had a farm - ee-i-ee-i-o.' ]
```

Do This:

Create a solution that allows users to:

1. Substitute individual words of the song with a different word of the user's choosing. If the user tries to replace a word that does not exist, print an error message and do not change the song.
2. Reverse the song so that the words (not letters!) are in the reverse order. No backmasking or subliminal messages from the Beatles, please ☺
3. Playback the song
4. Reset the song to the original version. Note that operations (a) and (b) above are persistent and the changes "stick" until you revert them. Therefore, you should be working with a COPY of the song as you're remixing it.
5. Load a new song from our playlist
6. When requested, show the title of the song you're currently remixing
7. Continue the above operations on demand, until the user explicitly quits your program.
8. Note that there is punctuation in some of the songs we've given you. **When you remix a song, you should remove the punctuation** (see the example reverse below). If you are unable to perform the remix, do NOT remove any punctuation - leave the song in its current state (see the last example below). Of course, if the song has already been altered, the "current state" has no punctuation anyway, so there's no punctuation alteration you would perform anyway.

We will be performing both unit-testing of your solution AND some whole-system tests as well. For unit tests, you must provide the following three functions in remix_master.py:

- `substitute(song: list, old_word: str, new_word: str ) -> bool:`
  This function takes a song (which is a list containing multiple strings as the example above

shows), an old word to replace and a new replacement word. It replaces every occurrence of the old word with the new word. If the replacement was successful (the old word was found and replaced) this function returns `True`, otherwise it returns `False`.
Any punctuation existing in the song should be stripped out IF this operation is successful. If the operation is not successful, the original list should not be altered.
NOTE: This function **is destructive**. It **mutates** the list passed in so your function must be sure to effect changes in the list.

- `reverse_it(song: list) -> list`
  This function takes a song (which is a list containing multiple strings) and reverses the song such that the words (not letters!) are in the reverse order (see requirement 2 above)
  NOTE: This function **is destructive**. Any punctuation existing in the song should be stripped out thereby **mutating** the list IF its operation is successful. If the operation is not successful, the original list should not be altered.

- `load_song(selection: int) -> list`
  This function takes an integer (which is an index into our playlist) and returns a list that contains the selected song AND a string which represents the song title from our playlist. For the list returned: the song must be placed in index 0 and the title must be placed in index 1. If the selection is not valid, this function returns an empty list.
  Important Note: This function is mapped to **the user-centered selection** operation, NOT our computer science index scheme. Therefore, `load_song(1)` retrieves the `FIRST` song in our playlist, not the second song.
  PRE: This function's input parameters must be integers. Input values are NOT guaranteed to be within the range of our playlist
  POST: If selection is found in the playlist, a list containing the song and the song title will be returned. Otherwise an empty list will be returned

Example screen captures are below. Find a more extensive demo movie here: **ReMix-Master**

```
ReMix-Master:
 L: Load a different song
 T: Title of current song
 S: Substitute a word
 P: Playback your song
 R: Reverse it!
 X: Reset to original song
 Q: Quit?
 Your choice: t
-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫
You are mixing the song: Old MacDonald
```

```
ReMix-Master:
 L: Load a different song
 T: Title of current song
 S: Substitute a word
 P: Playback your song
 R: Reverse it!
 X: Reset to original song
 Q: Quit?
 Your choice: s
```

```
What word do you want to replace in the exis
What new word do you want to use for the son

ReMix-Master:
 L: Load a different song
 T: Title of current song
 S: Substitute a word
 P: Playback your song
 R: Reverse it!
 X: Reset to original song
 Q: Quit?
 Your choice: r

ReMix-Master:
 L: Load a different song
 T: Title of current song
 S: Substitute a word
 P: Playback your song
 R: Reverse it!
 X: Reset to original song
 Q: Quit?
 Your choice: p
Turn up the 808's and drop the beat! Here's
ee-i-ee-i-o - farm a had macdonald young
ee-i-ee-i-o - cow a had he farm that on and
there moo moo a and here moo moo a with
moo moo a everywhere - moo a there - moo a h
ee-i-ee-i-o - farm a had macdonald young
-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫
```

```
ReMix-Master:
 L: Load a different song
 T: Title of current song
 S: Substitute a word
 P: Playback your song
 R: Reverse it!
 X: Reset to original song
 Q: Quit?
 Your choice: q
Bravo! Your Grammy Award is being ship
```

Note that if you try to substitute a new word for a song and the song cannot be remixed, you should leave the current version untouched.

```
 Your choice: s
What word do you want to replace in the exi
What new word do you want to use for the so
Sorry. I didn't find new in the existing so

ReMix-Master:
 L: Load a different song
 T: Title of current song
 S: Substitute a word
 P: Playback your song
 R: Reverse it!
 X: Reset to original song
 Q: Quit?
 Your choice: p
Turn up the 808's and drop the beat! Here's
old macdonald had a farm - ee-i-ee-i-o.
and on that farm he had a cow - ee-i-ee-i-o
with a moo moo here and a moo moo there
here a moo - there a moo - everywhere a moo
old macdonald had a farm - ee-i-ee-i-o.
-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫-♫

ReMix-Master:
 L: Load a different song
 T: Title of current song
 S: Substitute a word
```

Notes:

- For your development effort, I'm giving you 3 songs for my playlist. *Do NOT hard-code your solution based on these 3 songs!!!* During our testing of your solution, we will expand the playlist to more than the three original songs. Your solution should be flexible enough to handle

any sized playlist that has at least 1 song in it. So, if my test playlist has 7 songs in it, your solution should still work properly.

- Focus on good procedural decomposition where each of your functions are short (<15 lines of code, discounting comments) and doing 1 thing well *(NB: Your main() can be a bit longer than 15 lines, if your user menu is being handled there)*.
- No global variables (Global CONSTANTS are okay)
- Stretch-goal (try your best on this): Work to see if you can design your solution with a good "separation of concerns". See if you can construct your code for the user-interface to deal primarily with those aspects (e.g. getting input from the user and printing to the screen) and the other "business logic" to work without intermingling print statements, etc.

What to submit:

- Your solution in a file named: remix_master.py
- Include any other .py files you've written for your “helper” functions you’ve written, if you've put them in separate files. Documentation/comments in remix_master.py should indicate any additional helper files you’ve included