Due Mar 30 by 11:59pm **Points** 100 **Available** after Mar 20 at 1pm

Home

Resources

Homework 6: Dictionaries, Files, Exception Handling

Programming Component (100% of this HW)

Just two programs this week, and NO written.txt.
Focus on Files, Exception Handling and Dictionaries

Program 1: Emojini - 60% of your grade

This assignment should make you :-) LOL!

For this program, you'll be implementing a text transformation solution that substitutes English words into their appropriate text-based emojis. There are a few sets of emojis in use, the most common ones are "Western" style (prevalent in North America and Europe) and "Eastern" Kaomoji (used more in Japan and other APAC areas). Our solution will handle the transformation for both of these text-based emoji sets.

Files:

- emojini.py
- **Starter files** **emoji_directives.txt** **(https://northeastern.instructure.com/courses/102990/files/13111497/download?download_frd=1)** **emojis.txt** **(https://northeastern.instructure.com/courses/102990/files/13111439/download?download_frd=1)** **recommendation_letter_k_orig.txt** **(https://northeastern.instructure.com/courses/102990/files/13111433/download?download_frd=1)** **recommendation_letter.txt** **(https://northeastern.instructure.com/courses/102990/files/13111434/download?download_frd=1)** **supercool.txt** **(https://northeastern.instructure.com/courses/102990/files/13111504/download?download_frd=1)** **text_to_friend_k.txt** **(https://northeastern.instructure.com/courses/102990/files/13111438/download?download_frd=1)**

Task Overview:

You will be creating a solution to transform existing text into their text-based emoji representations, and vice versa. We are providing a text file with the mapping between English and the emoji character set you are to use for this assignment. Our emoji file uses metadata, so you will need to design your solution to handle that style of data representation.

Note: For this assignment, the first line in the data file is metadata. It describes the columns of data provided. Your program should be flexible enough to work EVEN IF WE GIVE AN ALTERNATE FILE with similar data, as long as the metadata is present.

The metadata line look like this: **METADATA ENGLISH WESTERN KAOMOJI**
The word **METADATA** indicates that this line is NOT traditional data. The next 3 (or more, if we add more emoji sets) columns describe the sequence of the emoji information in the rest of the file. Therefore, for each line of this font file, English words are in the first column, Western-style emojis are in the second column and Kaomoji-style emojis are in the third column.

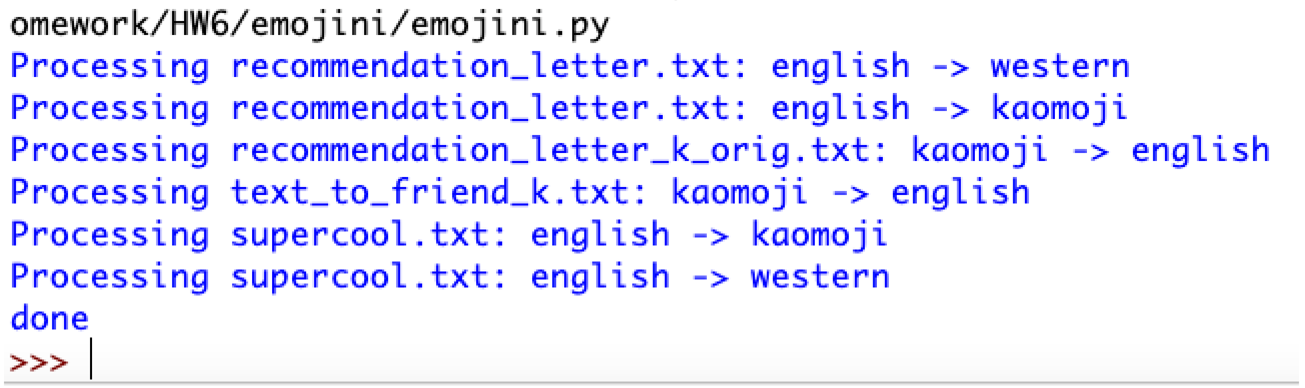
As part of your solution, you **must** provide the following function:

`batch_translate(emoji_file_name: str, directives_file_name: str)`
This function takes two strings: (1) the name of the emoji mapping file (we've given you one with emojis.txt) and (2) a directives file which gives instructions for the type of transformation to run,

and then orchestrates the process of converting the text as specified. Our autotester will call this function to test your solution, so whatever process needs to be initiated to perform the text transformation should be performed when this function is called.

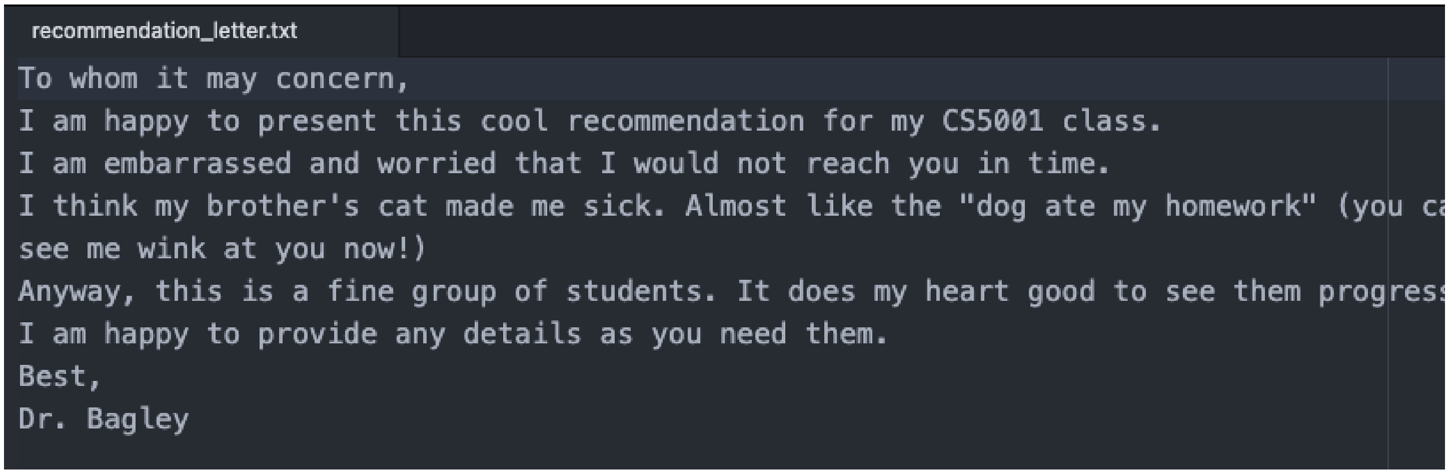
Do this:

- Write a program that translates English words to their Western or Kaomoji equivalents. Use the given emojis.txt as the basis for your translation. This is whole-word replacement, similar to what you accomplished with Homework 5. Unlike that homework, however, you should NOT be stripping punctuation from your finished transformation. We'll be using your transformation program as an automated responder to text messages and to create form letters (think mail-merge) and we need to retain the original punctuation in those messages.
- Your program does not require any user interface. Your output must be written to files. Minimal print-outs to the screen are allowed (and appreciated) to indicate that your file processing has completed, or that you've handled an exception of some type. However, we don't want to see paragraphs of text printed out, and certainly no input() from the keyboard.
  For example, here is our output after processing the directives file



```
omework/HW6/emojini/emojini.py
Processing recommendation_letter.txt: english -> western
Processing recommendation_letter.txt: english -> kaomoji
Processing recommendation_letter_k_orig.txt: kaomoji -> english
Processing text_to_friend_k.txt: kaomoji -> english
Processing supercool.txt: english -> kaomoji
Processing supercool.txt: english -> western
done
>>> 
```

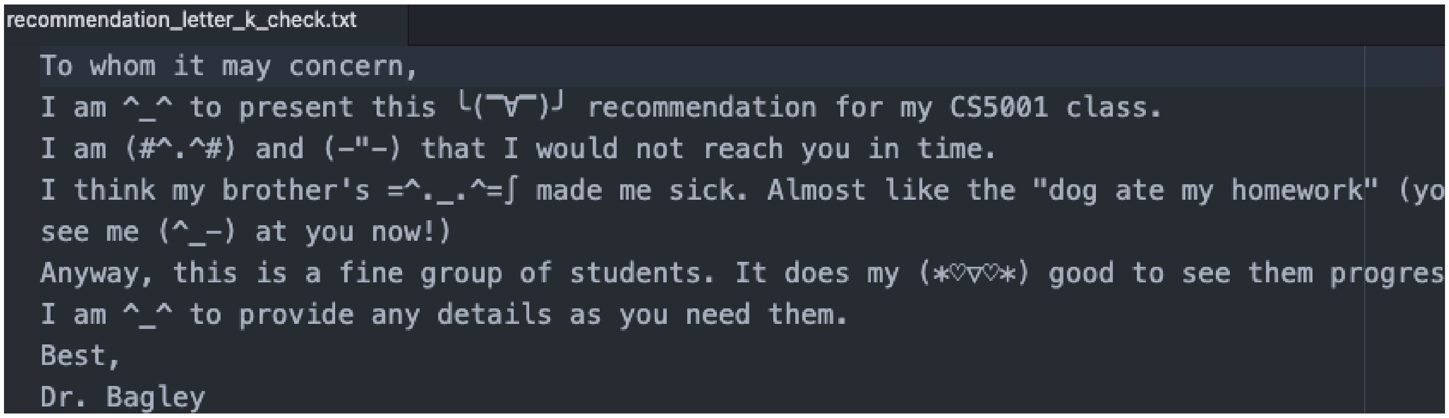
- Your program must read from a text file that gives instructions on the emoji substitution required, the input file to use as the source information AND the output file you should generate for your output. We've included one called emoji_directives.txt but your program should be able to handle ANY file we pass it when we call the appropriate function.
  - For example, the `emoji_directives.txt` starter file given gives you the transformations you must process in sequence. In one case we are asking for a transformation from English to kaomoji for recommendation_letter.txt. The source file: recommendation_letter.txt is given to you as a starter file. Your code must produce the output file recommendation_letter_k.txt



recommendation_letter.txt

```
To whom it may concern,
I am happy to present this cool recommendation for my CS5001 class.
I am embarrassed and worried that I would not reach you in time.
I think my brother's cat made me sick. Almost like the "dog ate my homework" (you ca
see me wink at you now!)
Anyway, this is a fine group of students. It does my heart good to see them progress
I am happy to provide any details as you need them.
Best,
Dr. Bagley
```

Source file



recommendation_letter_k_check.txt

```
To whom it may concern,
I am ^_^ to present this └(￣∀￣)┘ recommendation for my CS5001 class.
I am (#^.^#) and (-"-) that I would not reach you in time.
I think my brother's =^._.^=∫ made me sick. Almost like the "dog ate my homework" (yo
see me (^_-) at you now!)
Anyway, this is a fine group of students. It does my (*♡▽♡*) good to see them progres
I am ^_^ to provide any details as you need them.
Best,
Dr. Bagley
```

Output file after transformation is performed

DO NOT do this:

- **Do not hardcode** your solution based on the current file. We will use different directives files and different emoji font files during our testing
- **Do not** forget to handle any file exceptions using Python's exception handling mechanism (try/except) and/or through the use of "defensive programming". We may "accidentally" give you a locked or non-existent file to process in our directives.txt. Your code should be able to handle that situation gracefully

Note: This homework - in addition to giving you practice with files & exception handling (and possibly dictionaries, if you use them) - gives you a taste of what is called "Process Orchestration", batch processing, and data-driven programming. Essentially, the file emoji_directives.txt "drives" how the program behaves, so that the same code you've written may give us different outcomes when we simply change the incoming data files. In larger & more complex programs, a concept similar to our directives.txt may include business rules and other "process flow" directions that are run by complex "process orchestration engines". The approach is also used by smaller teams and practitioners by giving non-technical subject-matter-experts a means of building small situational applications without the assistance of development teams working on larger projects.

Program 2: Hyperspace BnB - 40% of your grade

You've recently discovered that your new apartment is at the crux of an interstellar wormhole. More than that, intergalactic aliens are lining up to visit Arth (aka "Earth") and want to pay you big-time credits to rent your living space. You're going to create a solution that manages your guest reservations and bookings on Hyperspace BnB so you can take advantage of this amazing opportunity.
Location, location, location!

Files:

- hyperspace_bnb.py
- **Starter files** **travelers.txt (https://northeastern.instructure.com/courses/102990/files/13111452/download?download_frd=1)** **requests.txt (https://northeastern.instructure.com/courses/102990/files/13111451/download?download_frd=1)**
- Additionally, your solution must produce a file called bookings.txt More detail on this below.

As part of your solution, you **must** provide the following two functions:

`load_travelers(travelers_file_name: str)`
This function takes a string that holds the name of the file which contains all of the galactic travelers, and returns a data structure of your choice (List, Dictionary). You are free to use whatever data structure you want BUT whatever data structure is returned from this function must be "plug compatible" with the function described below (process_requests). In other words, the output of this function must be a valid input to process_requests.

`process_requests(travelers: <<whatever data structure you chose>>, request_file_name: str)`
This function takes two parameters: (1) data structure produced by load_travelers and (2) a

bookings request file name. The bookings request file contains a set of booking request from our alien friends trying to grab a week at your apartment. If a request is successful, this function should orchestrate the reservation for our alien customer and (create if necessary) update a file named bookings.txt. If the week requested is already booked OR if the alien customer does not have enough credits, the bookings.txt file should not be modified.

BnB Fees: Charge 500 credits for each week booked.

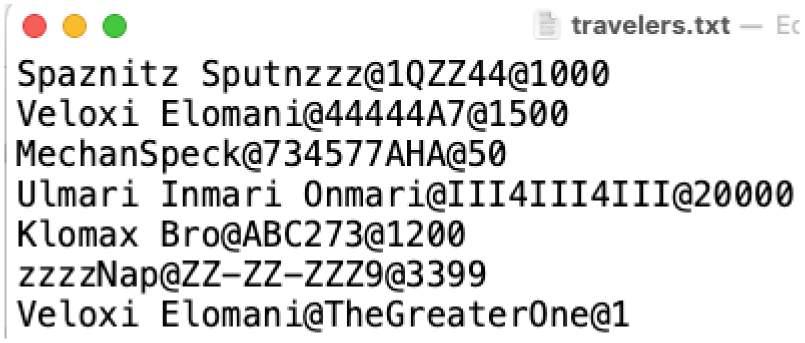
How does this all work?

First, understand that we will be calling your process_requests function like this:

```
process_requests(load_travelers("travelers_file_name.txt"),
"request_file_name.txt")
```

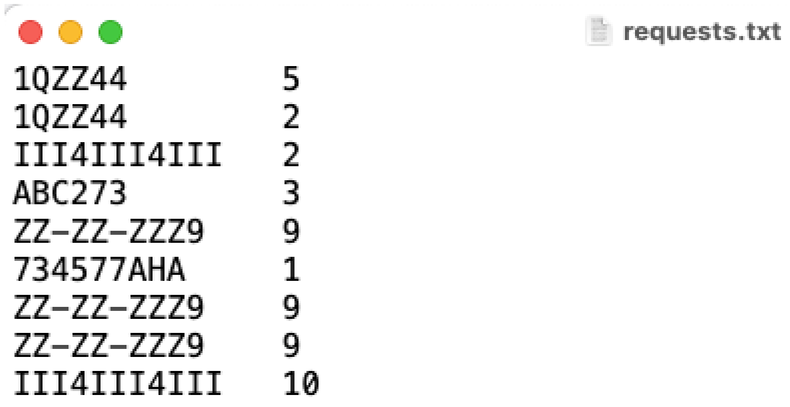
So again, the data representation you choose for your implementation is entirely your preference.

Here is a description of the data you'll encounter in our files:



```
Spaznitz Sputnzzz@1QZZ44@1000
Veloxi Elomani@44444A7@1500
MechanSpeck@734577AHA@50
Ulmari Inmari Onmari@III4III4III@20000
Klomax Bro@ABC273@1200
zzzzNap@ZZ-ZZ-ZZZ9@3399
Veloxi Elomani@TheGreaterOne@1
```

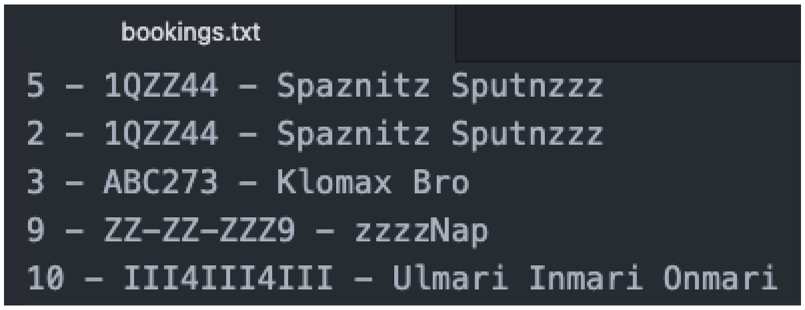
Travelers: Any travelers files we give you will be formatted like the above. The name of the customer is first, their user ID is second and the number of galactic credits in their account. Each field is separated by the @ sign, so you'll need to split the component pieces as you process the file



```
1QZZ44        5
1QZZ44        2
III4III4III   2
ABC273        3
ZZ-ZZ-ZZZ9    9
734577AHA     1
ZZ-ZZ-ZZZ9    9
ZZ-ZZ-ZZZ9    9
III4III4III   10
```

Requests: Request files have two columns separated by space. The first column is the customer ID (which is the same ID used in the travelers file) and the second column is the requested week the alien wants to book. You should process these requests in sequential order, first to last.

Therefore, if customer 1QZZ44 successfully books week 2, III4III4III will not be able to book the same week, even if they have sufficient funds to do so.


bookings.txt
```
5 - 1QZZ44 - Spaznitz Sputnzzz
2 - 1QZZ44 - Spaznitz Sputnzzz
3 - ABC273 - Klomax Bro
9 - ZZ-ZZ-ZZZ9 - zzzzNap
10 - III4III4III - Ulmari Inmari Onmari
```

The **bookings.txt** is the output file that your program will produce. It has 3 columns separated by " - ". The first column is the week that was booked, the second column is the customer ID and the third column is the customer name. The file you produce should be modeled after this format

Notes:

When you successfully book for a customer, you do NOT need to update the travelers.txt file to reflect the change in the alien's account balance. Our Home Office on Mars will handle the financial reconciliation on a nightly basis after we ship them your bookings file.
However, you DO need to keep track of the traveler's account balance in memory while your program is running. After you successfully book a week, ensure you deduct the galactic credits from the alien's account. You must prevent a week from being booked if an alien has insufficient funds!

No UI is required. Minimal output statements indicating your processing is complete is welcome so we know your program actually did what you promised, but extensive print statements are not appropriate. Here's our single output from our version of the program:

```
Finished processing reservations. Beam us up Scottie!
>>>
```

Do not forget to handle any file exceptions using Python's exception handling mechanism and/or through the use of "defensive programming". We will validate that your code can handle non-existent files gracefully.

Interesting Tidbit

This homework gives you some early experience with "plug compatibility". You own both the "source" and "consumer" of the data structure you'll use for travelers so your task is simplified and the function chaining match between `process_requests & load_travelers` "just works". But what if `load_travelers` was written by another team of developers, and they changed the data representation? To maintain plug-compatibility, we would need to create some "glue" code that we typically call "adaptors". Adaptors and pluggability are topics we'll cover more extensively next semester in OOD, but this is your gentle introduction with a bit of "hands on" practice.