# CS1026A Fall 2024
# Assignment 3: YouTube Emotions

**Important Notes:**

- Read the whole assignment document before you begin coding. This is a more complex specification than in past assignments and the examples and templates near the end of this document will be important in solving this assignment.
- Assignments are to be completed individually. Use of tools to generate code, working with another person, or copying from online resources are not allowed and will result in a zero on this assignment regardless of how much was copied.
- **A code template is given in Section 6 (on page 17) for your main.py and emotions.py files. We highly recommend using these as a starting point for your assignment. The code is also attached to the assignment on OWL.**

**Change Log:**

- **Nov. 4th:** The comments.csv file attached to Brightspace had an unexpected unicode character in one of the comments the changed the outcome of some of the examples given in this document. comments.csv has now been corrected and the examples in this document to match.

## 1. Learning Outcomes

By completing this assignment, you will gain skills relating to

- Functions
- Dictionaries and lists
- Complex data structures
- Text processing
- Working with TSV and CSV files
- File input and output
- Exceptions in Python
- Simple module use
- Writing code that adheres to a given specification
- Working with real world problem

## 2. Background

With the emergence of social media sites such as YouTube, Facebook, Reddit, Twitter (also known as X), LinkedIn, and WhatsApp, more and more data is being produced and made accessible online in a textual format. This textual data, such as YouTube comments, Tweets, or Facebook posts, can be hard to process but is incredibly important for organizations as it offers a current snapshot of the public's emotions (affinity) or sentiment about a topic at a current point in time. Having a live view of your customer's current affinity towards your products or the public's view of your political campaign can be critical for success.

Much work has been done towards the goal of creating large datasets of word affinity or sentiment. One such effort is the National Research Council (NRC) Emotion Lexicon which is a list of English words and their associations with eight basic emotions (anger, fear, anticipation, trust, surprise, sadness, joy, and disgust) and two sentiments (negative and positive).

Our goal in this assignment is to use a simplified version of the NRC Emotion Lexicon to classify YouTube comments based on one of the following emotions anger, joy, fear, trust, sadness, or anticipation. Based on the emotion contained in each comment for a particular video we then want to generate a report that details the most common emotions YouTube users have towards that video based on their comments.

## 3. Datasets

Your Python program will deal with two datasets, a keywords data set that contains a simplified version of the NRC Emotion Lexicon (this dataset will remain the same for all tests of your program) and a Comma-Separated Values (CSV) file that contains the comments for a particular YouTube video (this dataset will change for each test of your program).

### 3.1 Keywords Dataset (TSV File)

The **keywords.tsv** file attached to this assignment contains a simplified version of the NRC Emotion Lexicon. This is a Tab-Separated Values (TSV) file in which each line of the file contains a single word and its emotional classification based on six emotions (anger, joy, fear, trust, sadness, and anticipation). Each word in the file may be classified as having one or more emotions.

The following is an example of the first 10 lines of this file where tab (\t) characters are represented by arrows (→):

```
abacus→0→0→0→1→0→0
abandon→0→0→1→0→1→0
abandoned→1→0→1→0→1→0
abandonment→1→0→1→0→1→0
abbot→0→0→0→1→0→0
abduction→0→0→1→0→1→0
abhor→1→0→1→0→0→0
abhorrent→1→0→1→0→0→0
abolish→1→0→0→0→0→0
abominable→0→0→1→0→0→0
```

Each line starts with a keyword and is followed by a score (0 or 1) for each emotion in this order: anger, joy, fear, trust, sadness, and anticipation. If a 1 is present it means that keyword is related to that emotion. If a 0 is present the keyword is unrelated to that emotion.

For example, according to the above the word **“abacus”** is related to the emotion of **trust** and no other emotions. The word **“abandon”** is related to the emotions **fear** and **sadness** and no other emotions.

All words in the dataset will be related to at least one emotion. This file’s contents will remain the same for all tests but may be given a different filename based on the users input (e.g. it could be named keys.tsv or words.tsv rather than keywords.tsv).

## 3.2 Comments Dataset (CSV File)

The user will provide a Comma-Separated Values (CSV) file that contains a set of YouTube comments for a particular video. The name of this file will change based on the user’s input but will always end in .csv and have the same format.

The following is an example of a possible line from this file (the file may contain one or more lines). Note that this document wraps the line on to multiple lines but in the file this is one line ended by a line break (\n):

```
2,PixelPioneer24,brazil,The excavation scenes in the movie were
excellent but the unnecessary derision of the hero's motives seemed
unfair. His eventuality of success was not adequately showcased.
```

Each line of this file will contain four values separated by a single comma character (,). The values will always be in the following order:

```
Comment ID, Username, Country, Comment Text
```

Comment ID is a unique positive integer identifier for the comment. Username is the username of the user who posted the comment. Country is the user's home country, and comment text is the text the of the comment posted by the user.

No value will contain a line break or a comma character. The capitalization of country names could be different for each line even if it is for the same country, but the country will always be spelled the same.

Space characters will only occur in the comment text or country name.

## 4. Tasks

In this assignment, you will write two Python files, **emotions.py** and **main.py**, that will attempt to determine the most common emotion expressed in a YoutTube video's comments. You will create a number of functions (as specified in the Functional Specification in Section 5) that will perform simple sentiment analysis on the YouTube comments.

To accomplish this, you will need to do the following:

1. **Accept input from the user**: The user will specify the file names of the keywords and comments data sets as well as the name of the report file your program will create. The user will also input the name of the country they wish to filter the comments by.
2. **Read**. Your program will read in the keyword and comments datasets and store them in the formats described in the functional specification (in Section 5).
3. **Clean**. The text of the comments will be cleaned to remove any punctuation and convert them to all lowercase letters.
4. **Determine Emotion**. You will use the keyword's dataset to determine the overall emotion expressed in each comment.
5. **Generate Report**. Based on your analysis of each comment, you will create a report file that contains a summary of the most common emotion expressed as well as how common each emotion was (as specified in Section 5).

Additionally, you must follow the functional specification presented in Section 5 and the rules and requirements in Section 8.

# 5. Functional Specification

## 5.1 emotions.py

The functions described in this section should be present in your emotions.py file and must be used in some way in your program to read, clean, process, analyze, or report on the comments in the given dataset. Each function and its parameters must have the same name and spelling as specified below:

### `clean_text(comment)`

This function should have one parameter, comment, which is a string that contains the text of a single comment from the comments dataset. The function should clean this text by replacing any characters that are not letters (A to Z) and replacing them with space characters. It should also convert the comment's text to all lower case.

This function should return the cleaned text as a string.

**Example:**

```
clean_text("This4is-an example. It's a b*t silly.")
```

will result in this output:

```
this is an example  it s a b t silly
```

### `make_keyword_dict(keyword_file_name)`

This function should read the Tab-Separated Values (TSV) keywords file as described in Section 3.1. ***keyword_file_name*** is a string containing the name of the keywords file. This function can safely assume that this file exists, is in the current working directory, and is properly formatted. Checks on the file's existence will be done in the main.py file described later in this document.

The function should return a **dictionary** with keys for each word in the file and the values of this dictionary should be a new dictionary for each keyword that contains a value for each emotion (anger, joy, fear, trust, sadness, and anticipation).

**Example:**
Assuming that keywords.tsv contains the following three lines (where → is a tab character):

```
abacus→0→0→0→1→0→0
abandon→0→0→1→0→1→0
abandoned→1→0→1→0→1→0
```

then calling

```
make_keyword_dict("keywords.tsv")
```

should result in the following nested dictionary data structure:

```
{'abacus': {'anger': 0,
            'joy': 0,
            'fear': 0,
            'trust': 1,
            'sadness': 0,
            'anticipation': 0},
 'abandon': {'anger': 0,
             'joy': 0,
             'fear': 1,
             'trust': 0,
             'sadness': 1,
             'anticipation': 0},
 'abandoned': {'anger': 1,
               'joy': 0,
               'fear': 1,
               'trust': 0,
               'sadness': 1,
               'anticipation': 0}
}
```

Note that to pass the Gradescope tests this function must return a dictionary and not another collection such as a list, the keyword keys must be spelled exactly as listed in keywords.tsv, and the emotions must be spelled correctly and in lower case.

***Hint:*** *You may find a number of the Python string methods helpful when creating this function.*

`make_comments_list(filter_country, comments_file_name)`

This function should read the Comma-Separated Values (CSV) file as described in Section 3.2. ***comments_file_name*** is a string containing the name of the CSV file and ***filter_country*** is a string containing either a country name or the string "all". This function should read the CSV file and return a **list** containing only comments for the given country listed in ***filter_country*** (or all countries if the string "all" is given).

The list should contain one element for each comment in the file that matches the country in the filter (or all comments if "all" is given). Each element in the list should be a dictionary that contains a key for the Comment ID, Username, Country and Comment Text. The keys should be named 'comment_id', 'username', 'country', and 'text' respectively.

The comment text should be **stripped** of any leading and trailing whitespace.

**Example 1:**

Assuming that comments.csv only contains the following two lines (note that the line is wrapped in this document and in the .csv file this is only two lines):

```
1,RetroRealm77,united states,I was a bit disappointed with the
film's portrayal of childhood heroism. It felt like the classic
elements were just concealed under layers of unnecessary savagery
and violence.

2,PixelPioneer24,brazil,The excavation scenes in the movie were
excellent but the unnecessary derision of the hero's motives seemed
unfair. His eventuality of success was not adequately showcased.
```

then calling

```
make_comments_list("all", "comments.csv")
```

should result in the following nested list and dictionary data structure:

```
[ {'comment_id': 1,
  'username': 'RetroRealm77',
  'country': 'united states',
  'text': 'I was a bit disappointed with the film's portrayal of
childhood heroism. It felt like the classic elements were just
concealed under layers of unnecessary savagery and violence.'},
 {'comment_id': 2,
  'username': 'PixelPioneer24',
  'country': 'brazil',
  'text': 'The excavation scenes in the movie were excellent but
the unnecessary derision of the hero's motives seemed unfair. His
eventuality of success was not adequately showcased.'} ]
```

**Example 2:**

Given the same contents of comments.csv as in Example 1, if the following function call with the country name **brazil** was made:

```
make_comments_list("brazil", "comments.csv")
```

then the only element in the returned list would be:

```
[ {'comment_id': 2,
  'username': 'PixelPioneer24',
  'country': 'brazil',
  'text': 'The excavation scenes in the movie were excellent but
the unnecessary derision of the hero's motives seemed unfair. His
eventuality of success was not adequately showcased.'} ]
```

**Example 3:**

Given the same contents of comments.csv as in Example 1, if the function was called with a country name that was not present in the file such as:

```
make_comments_list("not a real country", "comments.csv")
```

then the resulting list would be empty:

```
[]
```

Note that to pass the Gradescope tests this function must return a list and not another collection such as a set or dictionary, the values of each list element must be a dictionary, and the keys used in that dictionary must match the spelling and lowercase capitalization given in this section.

## `classify_comment_emotion(comment, keywords)`

This function takes the text of a **comment** and the **keywords** dictionary created by the **make_keyword_dict** function as parameters and classifies the comment as one of the possible emotions (anger, joy, fear, trust, sadness, and anticipation), returning the emotion as a string.

A comment is classified by first cleaning the text (using the **clean_text** function) and then checking each word in the comment against the keywords dictionary. A total for each possible emotion should be kept with each word in the comment matching a keyword adding to the totals (based on the values in the **keywords** dictionary).

**Example:**

For the comment:

```
The excavation scenes in the movie were excellent but the
unnecessary derision of the hero's motives seemed unfair. His
eventuality of success was not adequately showcased.
```

the text should be first cleaned using **clean_text** to get:

```
the excavation scenes in the movie were excellent but the
unnecessary derision of the hero s motives seemed unfair  his
eventuality of success was not adequately showcased
```

then each word should be checked against the keywords dictionary and the totals for each emotion kept. Words not matching any words in the dictionary (shown in `black` above) do not add to the scores. For example, using the full **keywords.tsv** dataset the words shown in ***`blue`*** above have matches in the keyword dataset and would result in the following totals:

| Word | anger | joy | fear | trust | sadness | anticipation |
|---|---|---|---|---|---|---|
| excavation | 0 | 0 | 0 | 0 | 0 | 1 |
| excellent | 0 | 1 | 0 | 1 | 0 | 0 |
| derision | 1 | 0 | 0 | 0 | 0 | 0 |
| hero | 0 | 1 | 0 | 1 | 0 | 1 |
| unfair | 1 | 0 | 0 | 0 | 1 | 0 |
| eventuality | 0 | 0 | 1 | 0 | 0 | 1 |
| success | 0 | 1 | 0 | 0 | 0 | 1 |
| **Total:** | **2** | **3** | **1** | **2** | **1** | **4** |

Therefore, this comment would be classified as having the emotion of **anticipation** and the string *"anticipation"* should be returned by the function as it as the highest score.

In the event of a tie, the emotions should be given priority in this order: 1) anger, 2) joy, 3) fear, 4) trust, 5) sadness, and 6) anticipation.

***Hint:*** *You may find the string split method useful for looping through words rather than characters.*

### `make_report(comment_list, keywords, report_filename)`

This function takes the **comment_list** (created by the **make_comments_list** function), the **keywords** dictionary (created by the **make_keyword_dict** function), and a string containing the file name of the report to generate (**report_filename**) as parameters.

A new file should be created with the file name in **report_filename** and it should contain the name of the most common emotion classification in the **comment_list** dataset as well as a count of the number of comments classified as each emotion. In the event of a tie the emotions should be given priority in this order: 1) anger, 2) joy, 3) fear, 4) trust, 5) sadness, and 6) anticipation.

The format of the report should match the following example which is based on the attached comments.csv and keywords.tsv with a country filter of *"all"*:

```
Most common emotion: anger

Emotion Totals
anger: 6 (40.0%)
joy: 1 (6.67%)
fear: 1 (6.67%)
trust: 3 (20.0%)
sadness: 3 (20.0%)
anticipation: 1 (6.67%)
```

The emotion totals should occur in the same order (regardless of the counts) but the values would be different depending on the **comment_list** and **keywords** dictionary passed to the function.

All percentages should be rounded to two digits and all six emotions should always be listed even if their count is zero. **Important:** in your report file each percentage must be written with one or two decimal places. A value such as 20.000% or 6.6700% would be **wrong** even though it is technically rounded as there are too many decimal places. Your output must be formatted exactly as shown in the example above including the spacing and line breaks.

### Return

The function should **return** the name of the most common emotion; in this example it would be *"anger"*.

### Exception

In the event that the **comment_list** contains no comments (i.e. it is an empty list), the function should raise a RuntimeError containing the text "No comments in dataset!".

***Reminder:*** *The report should be saved to a file and not output to the screen or returned by the function. Only the name of the most common emotion should be returned.*

## 5.2 main.py

The program in main.py should ask the user for the file names of the keyword file and comments file that the data will be read from, as well as the name of the report file that will be created. It must use the functions defined in the emotions.py file to perform the tasks described in Section 4 and write the final report.

Your main.py file must contain the following two functions (**ask_user_for_input** and **main**) as specified:

### `ask_user_for_input()`

This function takes no parameters but asks the user to input the file names of the keywords TSV file, the comments CSV file, the country to filter by, and the file name of the report to be generated. These three filenames and the country name are returned in a tuple in this order: 1) keyword filename, 2) comment filename, 3) country name (converted to lower case), and 4) report filename.

**Example (of valid input):**

```
Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): Canada
Input the name of the report file (ending in .txt): report.txt
```

User input is shown in ***green*** and input prompts in `black`. Note that the filenames and country are based on the user's input and can not be hardcoded to one set value. This means that the filenames could be different depending on the values input by the user.

In this case the following tuple would be returned:

```
('keywords.tsv', 'comments.csv', 'canada', 'report.txt')
```

Note that the country name was converted to all lowercase.

**Exceptions**

Your ask_user_for_input() method must complete the following checks on the user input. If the input does not pass a check, an Exception should be raised causing the function to exit immediately.

Exceptions should be raised as soon as the invalid input is given. For example, if the keyword file does not exist, an exception should be raised before asking the user to input the comments file name.

**Check 1: File Extension**

For each of the three filenames, if the user inputs a filename ending in the wrong file extension (.csv, .tsv, or .txt) the function should raise a **ValueError** exception with a message stating that the file extension is incorrect such as *"Keyword file does not end in .tsv!"*. The text of this message must be **exactly** the following for each file:

- **Keyword File:** *"Keyword file does not end in .tsv!"*
- **Comments File:** *"Comments file does not end in .csv!"*
- **Report File:** *"Report file does not end in .txt!"*

**Check 2: Files Exist**

For the keyword and comment files you must check if the file exists using the os.path.exists function. If it does not, your function must raise a **IOError** exception with text explaining that the function does not exist. The message should have the text "<file name> does not exist!" where <file name> is replaced with the filename such as *"keywords.tsv does not exist!"*, where keywords.tsv is the missing file.

For the report file, if the file already exists, an **IOError** should be raised with text stating that "<file name> already exists!" where <file name> is the name of the report file. For example "report.txt already exists!" where the report file is named report.txt. This is to help prevent accidentally overwriting any files.

**Check 3: Valid Country**

Lastly you must check that the country input is either "all" or one of the following countries: 'bangladesh', 'brazil', 'canada', 'china', 'egypt', 'france', 'germany', 'india', 'iran', 'japan', 'mexico', 'nigeria', 'pakistan', 'russia', 'south korea', 'turkey', 'united kingdom', or 'united states'. If any other country or word is input, a **ValueError** should be raised with the text "<country> is not a valid country to filter by!" where <country> is the country the user input.

This subset of countries was chosen as they tend to occur in the datasets, we are using more than others. In more realistic scenario you would likely want to include all valid country names in this list, but this assignment limit to the above-mentioned countries. Keep in mind that this only limits the countries a user can filter by, it does not limit what country names can occur in the dataset.

### `main()`

This function handles calling the other functions in main.py and emotions.py to perform the tasks listed in Section 3. It should check for any exceptions being raised by the **ask_user_for_input** function, output the error message contained in the exception (this can be done by simply printing the exception with print()), and ask the user to input the values again if any exception is raised.

Once valid input has been received, it should call the functions from emotions.py required to analyze the comments and generate the report.

Lastly it should output to the screen the most common emotion in the comment data set. This should be displayed as "Most common emotion is: <emotion name>" where emotion name is the name of the emotion such as "Most common emotion is: anger" if the emotion is anger.

If the **make_report** function raises a RuntimeError exception (e.g. the comment list was empty), it should output the message contained in that error.

**Example 1:**

For the values in the attached keywords.tsv and comments.csv files:

```
Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): all
Input the name of the report file (ending in .txt): report.txt
Most common emotion is: anger
```

User input is shown in ***green*** and the contents of the outputted report.txt file is:

```
Most common emotion: anger

Emotion Totals
anger: 5 (33.33%)
```

```
joy: 2 (13.33%)
fear: 1 (6.67%)
trust: 3 (20.0%)
sadness: 3 (20.0%)
anticipation: 1 (6.67%)
```

**Example 2:**

For the same values in keywords.tsv and comments.csv but a country of “Canada”:

```
Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): Canada
Input the name of the report file (ending in .txt): report_cad.txt
Most common emotion is: sadness
```

And the contents of report_cad.txt would be:

```
Most common emotion: sadness

Emotion Totals
anger: 1 (16.67%)
joy: 0 (0.0%)
fear: 0 (0.0%)
trust: 2 (33.33%)
sadness: 3 (50.0%)
anticipation: 0 (0.0%)
```

**Example 3:**

In this example invalid inputs are given, and the user is asked to input them again.

```
Input keyword file (ending in .tsv): not_a_real_file.tsv
Error: not_a_real_file.tsv does not exist!

Input keyword file (ending in .tsv): real_file_wrong_extension.txt
Error: Keyword file does not end in .tsv!

Input keyword file (ending in .tsv): keys.tsv
Input comment file (ending in .csv): not_a_real_file.csv
Error: not_a_real_file.csv does not exist!
```

```
Input keyword file (ending in .tsv): keys.tsv
Input comment file (ending in .csv): bad_file_extension.tsv
Error: Comment file does not end in .csv!

Input keyword file (ending in .tsv): keys.tsv
Input comment file (ending in .csv): c.csv
Input a country to analyze (or "all" for all countries): Duck
Error: duck is not a valid country to filter by!

Input keyword file (ending in .tsv): keys.tsv
Input comment file (ending in .csv): c.csv
Input a country to analyze (or "all" for all countries): Belgium
Error: belgium is not a valid country to filter by!

Input keyword file (ending in .tsv): keys.tsv
Input comment file (ending in .csv): c.csv
Input a country to analyze (or "all" for all countries): FrAnCe
Input the name of the report file (ending in .txt): report.txt
Error: report.txt exists, the report file can not already exist!

Input keyword file (ending in .tsv): keys.tsv
Input comment file (ending in .csv): c.csv
Input a country to analyze (or "all" for all countries): FrAnCe
Input the name of the report file (ending in .txt): report_france.txt
Error: No comments in dataset!
```

Note that the above is one run of the program. It should keep asking for input again if an exception occurs in the **ask_user_for_input** function. Also note that in this example, keys.tsv and c.csv are valid files that exist and the file report.txt already exists. “Belgium” is not in the list of valid countries so it is rejected and “FrAnCe” is accepted despite it’s odd capitalization as the **ask_user_for_input** function should convert it to lowercase. In this example, the c.csv file contained no comments for France, so the exception “No comments in dataset!” was raised by **make_report** function.

## 6. Templates

This section gives some starter code you should use in your program. You may not alter the names of any function or the parameters the functions take (this includes adding or removing parameters). You may not import any libraries or modules not included in the template code and all code you add should be inside a function (adding code outside of a function may cause the Gradescope tests to fail). You may add additional helper functions as needed.

### emotions.py

```
# add a comment here with your name, email, and student number
# you can not add any import lines to this file
EMOTIONS = ['anger', 'joy', 'fear', 'trust', 'sadness', 'anticipation']


def clean_text(comment):
    # add your code here and remove the pass keyword on the next line
    pass


def make_keyword_dict(keyword_file_name):
    # add your code here and remove the pass keyword on the next line
    pass


def classify_comment_emotion(comment, keywords):
    # add your code here and remove the pass keyword on the next line
    pass


def make_comments_list(filter_country, comments_file_name):
    # add your code here and remove the pass keyword on the next line
    pass


def make_report(comment_list, keywords, report_filename):
    # add your code here and remove the pass keyword on the next line
    pass
```

### main.py

```
# add a comment here with your name, email, and student number.
# do not add any additional import lines to this file.

import os.path
from emotions import *
```

```
VALID_COUNTRIES = ['bangladesh', 'brazil', 'canada', 'china', 'egypt',
                   'france', 'germany', 'india', 'iran', 'japan', 'mexico',
                   'nigeria', 'pakistan', 'russia', 'south korea', 'turkey',
                   'united kingdom',  'united states']


def ask_user_for_input():
    # add your code here and remove the pass keyword on the next line
    pass


def main():
    # add your code here and remove the pass keyword on the next line
    pass


if __name__ == "__main__":
    main()
```

### Note About Imports

It is important to import the files in the correct order and from the correct files. Main.py should import emotions.py as shown in the template above and not the other way around.

## 7. Extra Example

The files **keywords.tsv** and **comments.csv** should be attached to this assignment on OWL. The result of running them with the following countries is given below:

**Example 1: Country of "All"**

**Input/Output:**

```
Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): all
Input the name of the report file (ending in .txt): my_report.txt
Most common emotion is: anger
```

**Contents of my_report.txt:**

```
Most common emotion: anger

Emotion Totals
anger: 5 (33.33%)
joy: 2 (13.33%)
fear: 1 (6.67%)
trust: 3 (20.0%)
sadness: 3 (20.0%)
anticipation: 1 (6.67%)
```

**Example 2: Country of "brazil"**

**Input/Output:**

```
Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): brazil
Input the name of the report file (ending in .txt): report_brazil.txt
Most common emotion is: fear
```

**Contents of report_brazil.txt:**

```
Most common emotion: fear

Emotion Totals
anger: 0 (0.0%)
joy: 0 (0.0%)
fear: 1 (50.0%)
trust: 0 (0.0%)
sadness: 0 (0.0%)
anticipation: 1 (50.0%)
```

**Example 3: Country of "germany" (there are no comments for this country in the data set)**

```
Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): germany
Input the name of the report file (ending in .txt): report.txt
Error:  No comments in dataset!
```

**Example 4: Invalid Inputs (these files do not exist or have the wrong extension)**

```
Input keyword file (ending in .tsv): badfile.pizza
Error: Keyword file does not end in .tsv!

Input keyword file (ending in .tsv): this_file_does_not_exist.tsv
Error: this_file_does_not_exist.tsv does not exist!

Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): badcsvfile.duck
Error: Comment file does not end in .csv!

Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): not_a_real_csv_file.csv
Error: not_a_real_csv_file.csv does not exist!

Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): not_a_real_country
Error: not_a_real_country is not a valid country to filter by!

Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): JaPaN
Input the name of the report file (ending in .txt): badreportfile.exe
Error: Report file does not end with .txt!

Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): JAPAN
Input the name of the report file (ending in .txt): already_exists.txt
Error: already_exists.txt exists, the report file can not already exist!

Input keyword file (ending in .tsv): keywords.tsv
Input comment file (ending in .csv): comments.csv
Input a country to analyze (or "all" for all countries): jApAn
Input the name of the report file (ending in .txt): new_report_file.txt
Most common emotion is: anger
```

## 8. Non-Functional Specification

In addition to the other tasks and specifications given in this document, your program must also fulfill the following requirements:

1. Your code must be written for Python 3.10 or newer.
2. You may **not** use any modules or third-party libraries not described in this document. Standard built-in functions such as the String, file, and math functions are fine. You should not have to import anything other than your **emotions.py** and the **os.path** module. TAs may manually remove marks from your Gradescope test if you violate this rule.
3. You must document your code with brief comments. Each file should contain a comment at the top of the file with your name, western e-mail, student number, and a brief description of what is contained in that file. At least one comment should also be given for each function that describes its purpose, parameters, and values returned. You should also include any additional comments to document any lines that may be unclear to the reader.
4. Your program must be efficient and terminate within a reasonable time limit. All gradescope test cases (including any hidden cases) must terminate within the autograder's time limit.
5. Assignments are to be done individually and must be your own original work. You may not show or otherwise share your code for this assignment with others or use tools to generate your code for you. Software will be used to detect academic dishonesty (cheating). If you have any questions about what is or is not academic dishonesty, please consult the document on academic dishonesty and ask any questions to your course instructor before submitting this assignment.
6. You must follow Python style and coding conventions and good programming techniques, for example:
    a. Meaningful variable and function names.
    b. Use a consistent convention for naming variables, constants, and functions (snake case is recommended).
    c. Readability: indentation, white space, consistency.
7. All of your code should be contained in the files **main.py** and **emotions.py**. Only submit these files and no others and ensure the filenames match exactly. It is your responsibility to ensure you have submitted the correct files.
8. All function names and outputs should follow the specifications given in this document exactly. Not following the specifications may lead to test cases failing. It

is your responsibility to ensure you have followed them correctly and your tests are passing.

9. Hardcoding or any other attempt to fool Gradescope's autograder will result in a zero grade for that test being manually assigned and possibly additional penalties. If you have any doubts about what is or is not hardcoding, please ask your instructor by posting to the course forums.
10. Frequently backup your work remotely (e.g. using OneDrive) in a way that is secure and private. No extension will be given for lost or corrupted files or submitting incorrect files.

# 9. Marking and Submission

## 9.1 Submission

You must submit the 2 files (main.py and emotions.py) to the Assignment 3 submission page on Gradescope. There are several tests that will automatically run when you upload your files. The result of each test with be displayed to you, but not necessarily the exact details of the test. **It is your responsibility to ensure all tests pass before the assignment due date.**

It is recommended that you create your own test cases to check that the code is working properly for a multitude of different scenarios (some example datasets have been provided for you with this document on OWL).

Assignments will not be accepted by email or by any other form other than a Gradescope submission.

## 9.2 Marking

The assignment will be marked as a combination of your auto-graded tests (both visible and hidden tests) and manual grading of your code logic, comments, formatting, style, etc.

**Marks will be deducted for failing to follow any of the specifications in this document** (both functional and nonfunctional), not documenting your code with comments, using poor formatting or style, hardcoding, or naming your files incorrectly.

**Marking Scheme:**

- Autograded Tests: **24.5 points**
- Header comment including your name, student ID, course info, creation date, and description of file: **1.5 points**
- Descriptive in-line comments throughout code: **1 point**
- Meaningful variable names: **1 point**

Total: **28 points**

## 9.3 Late Submissions

Late assignments will only be accepted up to 3 days late and only if you have enough late coupons remaining (at least one for each day late). If you submit one day late, you will need to use 1 late coupon. 2 days late, 2 late coupons. And 3 days late, 3 late coupons. If you have insufficient late coupons remaining or submit more than 3 days late, you will receive a zero grade on this assignment.

It is your responsibility to track your late coupon use. Any values shown on OWL should be considered an estimate and may not be accurate or up to date.

Unlimited resubmissions are allowed, but the late penalty will be determined by the date/time of your most recent (last) resubmission. This means if you resubmit past the deadline, your assignment will be considered late.