A Database of Scores for Your Favorite Video Games or Movies

For this PA, you'll write two programs, each with two functions. **It isn't sufficient for your programs to run correctly; 25% of your grade will be based on the following:**

- **use of headers with required information for both programs (5%),**
- **use of a docstring in each function (5%),**
- **use of comments,**
- **choice of lvalue and variable names, and**
- **readability, i.e., include whitespace (15% for last three).**

You'll need to submit your programs to Canvas as a single zipped file so if you don't know how to zip a file, ask your TA during your lab or else google how to do it for your operating system. *Note that the grading rubric for this PA is available in Canvas.*

Header information. Each program must include the following information in the header.

- Name
- Date
- CptS 111, Spring 2023
- Programming Assignment #5a or #5b
- Name of program
- Brief description and/or purpose of the program; include sources

Background Information

In this PA you're going to write two programs. The first program will be used to create a database of your favorite movies or video games (or you can create two databases and do both!). For each movie or video game title, you'll score seven different attributes as high (5), medium (3), or low (1). For the second program, a numerical score between 7 and 35 will be displayed for a title based on how you scored it. The attributes you'll score are Visuals, Production Design (overall look and feel), Music, Sound Design (overall sound including sounds and music), Story, Characters, and Gameplay (for video games) or genre (for movies, assuming you like some types better than others).

Program Requirements

Now let's turn to the functions you need to code for each program.

Program 5a

For the first program, you're going to create or add to a dictionary database of your favorite movies or video games (choose one).

- **`add_faves()`**: A void function with one parameter, the dictionary that you opened or initialized in your `main()` function. What you need to do to code this function:
  - Write the function header.
  - Open a file called `faves.txt` with the `'w'` option (write to file). Assign it to an lvalue with an appropriate name:
    `file_lvalue = open('faves.txt', 'w')`
    where `file_lvalue` is a dummy name representing the name you chose for your output file. Note that `faves.txt` is the name of the file that will appear in a folder in your computer, and `file_lvalue` is the name of the file that represents `faves.txt` that you use in your program. It's only used in your code and won't appear in a folder in your computer! This can be a little confusing, so you may need to think about it. It's important to understand the difference.
  - Write a `while`-loop header that will continue the iterations forever, e.g., `while True:` or `while 'Go, Cougs!':`, (don't worry; you'll break out of the loop; also, recall that almost everything in Python evaluates to `True`). Each time the loop is executed, one title and one list of scores is added to your dictionary. The title is a dictionary key, and the list of scores is its value (recall that dictionaries are made up of key-value pairs). In the body of the `while`-loop, do the following:
    1. Print a blank line (see examples below).
    2. Initialize an empty list for the scores with an appropriate name.
    3. If you're creating a dictionary of your favorite video games, prompt for a video game title; if you're creating a dictionary of your favorite movies, prompt for a movie title (**see examples below for proper text prompts**). Assign the title returned by the `input()` function to an lvalue with an appropriate name.
    4. Test to see whether the title input is empty, i.e., nothing (`''`) was entered, using a *simple* conditional. If true, then use the `break` command to break out of the `while`-loop and, thus, end the addition of more dictionary entries, i.e., key-value pairs.
    5. Prompt for seven different scores (see examples below), one at a time.
    6. Append each score to the list you initialized in 2 above, but only use the first character of the input and also make sure it's lowercase. To accomplish this, use the following as the argument of the `.append()` method:
       `score_lvalue[0].lower()`
       where `score_lvalue` is a dummy name for each of the seven names you chose when doing 5 above. `score_lvalue[0]` gives the first character in the string, and the `.lower()` method ensures this character is lowercase.
    7. Use the following statement to clean up the title:
       `title_lvalue = title_lvalue.lower().strip()`
       where `title_lvalue` is a dummy name for the name you chose in 3, and the `.strip()` method removes whitespace from both sides of `title_lvalue`. Note that Python executes methods from left to right.

8. A title you enter each time the loop is executed is a new key and the list of scores for this title is its value. Use the correct command to add this key-value pair to your dictionary. **This is the final statement inside the `while`-loop!!!**

- Print the entire dictionary (the parameter you passed to the function) to the output file `file_lvalue`.
- Close the output file using `file_lvalue.close()`. Recall that `file_lvalue` is a dummy name representing the name you chose earlier.
- Add a `return` statement.
- Add a docstring.

- **`main()`**: A void function with no parameters. This function prompts to see whether you already have a favorites database (dictionary). To determine this, use a conditional (see examples below). If you don't have a favorites database, you must initialize an empty dictionary and then call `add_faves()` with the dictionary name as its parameter; if you do have one, you must 1) open the dictionary file, 2) read it, and 3) call `add_faves` with the dictionary name as its parameter. What you need to do to code this function:
  - Write the function header.
  - Use the `input()` function to prompt to see whether a database (dictionary) already exists as shown in the examples below.
  - Next use the following statement:
    ```
    input_lvalue = input_lvalue.lower().strip()
    ```
    where `input_lvalue` is a dummy name for the lvalue you assigned to the value returned by the `input()` function.
  - Code an `if-elif-else` construct that tests to see if i) `input_lvalue` is `'n'`, ii) tests to see whether `input_lvalue` is `'y'`, or else iii) prints a message as shown in one of the examples below. If `'n'`, then initialize an empty dictionary called `dict_lvalue`, where `dict_lvalue` is a dummy name for the name you chose, and pass this to `add_faves()`. If `'y'`, use the following code:
    ```
    file_lvalue = open('faves.txt', 'r').read()
    dict_lvalue = eval(file_lvalue)
    ```
    where `file_lvalue` and `dict_lvalue` are again dummy names for whatever names you chose. However, it's best if `dict_lvalue` has the same name as the one you chose when you initialized the empty dictionary. Call `add_faves()` using this name as the argument.
  - Add a docstring.
  - Note that in Python `return` isn't used in `main()`. This is not necessarily the case in C.

Examples for first program (input in bold). Note where the blank line is printed!

```
Do you already have a faves.txt file [y/n]? y

```

```
Enter movie title [return to end]: Nomadland
Visuals [hi/med/lo]: hi
Production Design [hi/med/lo]: hi
Music [hi/med/lo]: med
Sound Design [hi/med/lo]: med
Story [hi/med/lo]: hi
Characters [hi/med/lo]: hi
Genre [hi/med/lo]: med

Enter movie title [return to end]:<Return>
```

```
Do you already have a faves.txt file [y/n]? n

Enter video game title [return to end]: Grand Theft Auto
Visuals [hi/med/lo]: hi
Production Design [hi/med/lo]: hi
Music [hi/med/lo]: med
Sound Design [hi/med/lo]: hi
Story [hi/med/lo]: hi
Characters [hi/med/lo]: med
Game Play [hi/med/lo]: hi

Enter video game title [return to end]:<Return>
```

```
Do you already have a faves.txt file [y/n]? p
Problem with answer.  Please try again.
```

The result of your first program should be a file on the hard drive of your laptop or desktop called **`faves.txt`**. Look for this file, and look at its contents. If, for example, you chose to create a dictionary database of your favorite movies, it should look similar to the following (although, hopefully, with more movie titles):

```
{'nomadland': ['h', 'h', 'm', 'm', 'h', 'h', 'm'], 'black panther':
  ['h', 'h', 'h', 'h', 'h', 'h', 'p']}
```

Note that the value for the key `black panther` is incorrect because of the last entry in the list. This is intentional because it's used as an example for Program 5b. However, you can always edit a dictionary using a text editor (not a word processor) to fix any errors.

Program 5b

Now let's turn to the second program. This one is used to query the database and return a score if the title is found. Note that it can be written with many more capabilities, but we'll keep it simple.

- **`score_faves()`**: A non-void function with one parameter, a list of scores, that returns the sum of the scores in the list. What you need to do to code this function:

- Write the function header.
- Initialize an accumulator for the total score.
- Code an **iterating** `for`-loop for the list of scores.
- Inside the loop, code an `if-elif-else` construct such that:
  - If `loop_var` is `'h'`, add 5 to the total score,
  - else if it's `'m'`, add 3 to the total score,
  - else if it's `'l'`, add 1 to the total score,
  - else print the message shown in the examples below and return **`'incalculable'`** to the calling program `main()`. **This return statement should be inside the body of the conditional.**

  where `loop_var` is a dummy name for the loop variable name you chose.
- Outside the loop, return the total score to the calling program.
- Add a docstring.

- **`main()`**: A void function with no parameters. This function prompts the user for the title they want to score and calls `score_faves()` to calculate the score. What you need to do to code this function:
  - Write the function header.
  - Use the following statements:
    ```
    file_lvalue = open('faves.txt', 'r').read()
    dict_lvalue = eval(file_lvalue)
    ```
    where `file_lvalue` and `dict_lvalue` are dummy names for the names you chose.
  - Prompt the user for the title they want to score (see examples below for proper text prompts to use).
  - Next use the following statement:
    ```
    title_lvalue = title_lvalue.lower().strip()
    ```
    where `title_lvalue` is again a dummy name for the name you chose for the lvalue when you prompted the user for a title.
  - Print a blank line.
  - Code an `if-else` construct that tests to see whether `title_lvalue` is in `dict_lvalue` (for the test statement, simply use the `in` command as explained in the last lecture for Ch. 3. If it isn't, then your code should print the message shown in one of the examples below. If it is, then your code should call `score_faves()` and print the results using the following statements:

```
score_lvalue = score_faves(dict_lvalue[title_lvalue])
print(f'Your score for {title_lvalue.title()} is {score_lvalue}.')
```

    where `score_lvalue` is a dummy name for the name you chose for the total score returned by `score_faves()`, and the `.title()` method capitalizes all the words in `title_lvalue`. To test to see whether a title is in a dictionary, use the construct: `if title_lvalue in dict_lvalue:`.

- Add a docstring.

Examples for second program (input in bold).

```
Enter title you want to score: Incredibles 2

Movie/Video game not in database [or typo in name].
```

```
Enter title you want to score: Black Panther

Problem with scores.
Your score for Black Panther is incalculable.
```

```
Enter title you want to score: Nomadland

Your score for Nomadland is 29.
```

Submission information. Use Canvas to submit your programs. Submit both in a single zipped file called `<first_initial+lastname>_pa5.zip`. If you don't know how to zip a file, ask your TA or else google how to do it for your operating system (Windows, macOS, or Linux).