Lab 3

Dictionaries, file handling, and error handling

3.1 Learning goals

- You should be able to work with dictionaries.
- You should be able to read and write data from files.
- You should be able to raise and handle errors.

3.2 Submission

The lab should be submitted as usual via PeerGrade. Don't forget to peer-grade after the submission deadline!

Remember: it's always a good idea to add a comment where it would help explain the purpose of a line or code block to someone reading your code. Likewise, document your functions so that a reader can easily understand what inputs they take and what the function does.

You may not use functions from any library in the lab, i.e. you should not use the "import" keyword anywhere in your solution. Submit all your code in a single file as in previous labs: functions first, then the main program that calls the functions.

Remember to remove your name from your code if for example Spyder adds it to your file.

3.3 Tasks

The goal of this lab is to create a program that loads information about a music library from a file and helps the user create playlists from their library.

Our music library will be text files in a certain format. A typical music library might look like this (`80s_library.txt` in the "Code for labs" file on the course site):

```
Kate Bush,Get Out of My House,5:51
Kate Bush,Rocket's Tail,4:07
```

```
Talking Heads,Burning Down The House,4:01
Talking Heads,Houses in Motion,4:33
Talking Heads,Moon Rocks,5:45
```

A library consists of several lines of text, each of which has an artist name, song name, and song length separated by commas (`,`). The length of a song is stored as a number of minutes followed by a number of seconds, separated by a colon (`:`).

Note: This is not a perfect format for storing a music library—how would we store a song with a comma in its name? Most "comma-separated value" data formats have ways to get around this issue (see Wikipedia or the Python `csv` library), but we will keep things simple for this lab.

3.3.1 Task 1: Parsing and printing song lengths (1 point)

Our data format stores song lengths in a human-readable string format, but if we want to calculate with song lengths, it will be more useful to store a length as a single integer, the total number of seconds. Here is a function that converts an integer to a string using Python's *f-strings*:

```
def seconds_to_string(seconds):
    return f"{seconds // 60}:{seconds % 60:02d}"
```

For example, `seconds_to_string(101)` returns the string `'1:41'`. Write a function `string_to_seconds(s)` that does the opposite conversion: given a string in the minute-second format, it should return the total number of seconds as an integer. If given a string *not* in this format, it should instead raise a `ValueError`. A minute-second string is correctly formatted when it consists of a non-negative integer followed by a colon (`:`) followed by two digits that make a non-negative integer less than `60`.

Sample tests:

```
>>> string_to_seconds('1:41')
101
>>> string_to_seconds('207:53')
12473
>>> string_to_seconds('0:00')
0
>>> string_to_seconds('4:5')
Traceback (most recent call last):
  ...
ValueError
>>> string_to_seconds('2:70')
Traceback (most recent call last):
  ...
ValueError
>>> string_to_seconds('hej')
Traceback (most recent call last):
  ...
ValueError
```

3.3.2 Task 2: Loading a library from a file (4 points)

When we load a library into Python, we will store it in hierarchical, dictionary-based format. A library will be represented as a dictionary with artist names as keys. The value of the dictionary at an artist will

be *another* dictionary with that artist's songs as keys and their lengths (as integers) as values.

a) Write a function `read_library` that takes a filename string as an argument, reads the library stored in that file, and converts it to a dictionary of the form described above. (2 points)

Your function should have the following behavior in exceptional cases:

- If the file does not exist, the function should raise a `FileNotFoundError`.
- If the input file is malformed (if some line does not have 3 comma-separated entries or the time is not in minutes-second format), then the function should raise a `ValueError`.

You can assume that the same song does not occur twice in the library file.

Sample test:

```
>>> read_library('80s_library.txt')
{'Talking Heads': {
  'Burning Down The House': 241,
  'Houses in Motion': 273,
  'Moon Rocks': 345
  },
 'Kate Bush': {
  'Get Out of My House': 351,
  "Rocket's Tail": 247}
}
```

Note: you can assume that the input file ends in a newline (`'\n'`) character.

b) Write a function `print_library(lib)` that takes a library in dictionary format and *prints* its contents organized by artist. In addition to individual song information, your function should print out the total number of songs and total length for each artist and for the entire library, as shown in the example below. (2 points)

Sample execution:

```
>>> lib = read_library('80s_library.txt')
>>> print_library(lib)
Talking Heads (3 songs, 14:19)
- Burning Down The House (4:01)
- Houses in Motion (4:33)
- Moon Rocks (5:45)
Kate Bush (2 songs, 9:58)
- Get Out of My House (5:51)
- Rocket's Tail (4:07)
Total: 5 songs, 24:17
```

3.3.3 Task 3: Creating playlists (3 points)

Next, we'll write a function to create a playlist from a music library based on a theme (such as `House` or `Rock` music).

First, we need to decide on a format for playlists. Unlike a library, a playlist is meant to be played in order, so we'll represent it as a list of tuples. Each tuple will have three elements: one for the artist, one for the song, and one for the song's length (as an integer). Here's an example of a playlist:

```
[('Talking Heads', 'Houses in Motion', 273),
 ('Talking Heads', 'Moon Rocks', 345)]
```

a) Write a function `make_playlist(library, theme)` that takes a library (in dictionary form) and a "theme" string. Your function return a playlist consisting of all songs in the library whose names contain the theme as a substring. If there are *no* songs matching the theme, `make_playlist` should raise a `ValueError`. (2 points)

Sample execution:

```
>>> lib = read_library('80s_library.txt')
>>> make_playlist(lib, "House")
[('Talking Heads', 'Burning Down The House', 241),
 ('Talking Heads', 'Houses in Motion', 273),
 ('Kate Bush', 'Get Out of My House', 351)]
>>> make_playlist(lib, "Rock")
[('Talking Heads', 'Moon Rocks', 345),
 ('Kate Bush', "Rocket's Tail", 247)]
>>> make_playlist(lib, "Bluegrass")
Traceback (most recent call last):
  ...
ValueError
```

Note: The order of songs in your playlists does not need to match the examples above.

Tip: You can use the `in` operator to test whether one string is a substring of another. For example, `'el' in 'hello'` returns `True`, while and `'hej' in 'hello'` and `'hl' in 'hello` return `False`.

b) Write a function `write_playlist(playlist,filename)` that takes a playlist and a filename as arguments and saves the playlist in the given file, using the same comma-separated format as we use for libraries. (1 point)

For example, after running

```
>>> lib = read_library('80s_library.txt')
>>> house = make_playlist(lib, "House")
>>> write_playlist(house, "house_music.txt")
```

the file `house_music.txt` should contain the following text:

```
Talking Heads,Burning Down The House,4:01
Talking Heads,Houses in Motion,4:33
Kate Bush,Get Out of My House,5:51
```

3.3.4 Task 4: Putting it all together (2 points)

For your final task, we'll combine what we've done so far into a complete program. Write a function `main()` that

1. asks the user for a music library filename;
2. prints out the contents of the library;
3. asks the user for a playlist theme;
4. asks the user for a filename to save the playlist;
5. saves the playlist for the theme in that file.

Your function should handle the following exceptional situations:

- if the music library filename given by the user does not exist or the file is malformed, it should ask for a different filename.
- if there are no songs matching the theme, it should ask the user for a different theme.

Sample execution:

```
>>> main()
Which music library do you want to load? nonexistent.txt
That file does not exist.
Which music library do you want to load? 80s_library.txt
Talking Heads (3 songs, 14:19)
- Burning Down The House (4:01)
- Houses in Motion (4:33)
- Moon Rocks (5:45)
Kate Bush (2 songs, 9:58)
- Get Out of My House (5:51)
- Rocket's Tail (4:07)
Total: 5 songs, 24:17
Enter a playlist theme: Pythonwave
No songs match this theme.
Enter a playlist theme: Rock
Where do you want to save the playlist? rock_music.txt
Saved. Goodbye!
```

After this execution, the file `rock_music.txt` should contain the following text:

```
Talking Heads,Moon Rocks,5:45
Kate Bush,Rocket's Tail,4:07
```

3.3.5 Task 5 (0 points)

Read through, clean up, and document your code. In order that grading be objective, you should not include your name in the file you hand in.

Tips: read through "Basic principles of programming" under Resources on the course site for recommendations on how to write good code.