Title: Music Library Algorithm

Author: Shaun Clarke

Goal: This program mimics some of the basic functionalities of a music library.

Steps:

1. Import *Dict, List, and Union* from the *typing* module.
    a. This will be for type hinting
2. Define a class User:
    a. This parent class creates a user and their music collection.
    b. Define a class variable __users: Dict, which holds a dictionary.
    c. The construct or takes the user's first and last name and initializes the following.
        i. Generates a username
        ii. The user collection dictionary
        iii. A list of numbers to manage duplicates
        iv. register_user() that handles duplicate names
    d. Define a method register_user(self) -> None:
        i. This method registers a user and handles duplicate usernames.
        ii. Adds the username if it does not exist and return 0.
        iii. If the username exists, add a digit to the end of it.
    e. Define a method change_user(self, username: str) -> Union[object,str]:
        i. This method uses the username parameter to get a user from the __users dictionary.
    f. Define a method get_username(self) -> str:
        i. This method returns the username that was generated.
    g. Define a method get_music_collection(self) -> Dict:
        i. This method returns the user's music collection
    h. Define a method get_users(self) -> Dict:
        i. This method returns the class dictionary with all the users.

3. Define a class MusicUser(User):
    a. This child class inherits from the User parent class and handles the user actions.
    b. Define a constructor __init__(self, first_name: str, last_name: str):
        i. Call the parent class constructor super().__init__(first_name, last_name)
    c. Define a private method __is_collection_empty(self) -> bool:

        i. This method checks if the library is empty

d. Define a method is_song_in_library(self, title: str) -> bool:
    i. This method uses the title to check if a song exists in the library.
e. Define a method add_music_to_library(self, title: str, artist: str, genre: str) -> str:
    i. This method adds a song to the library.
    ii. If song exists
        1. Return exists
    iii. If not then use title, artist, genre to add song to library
    iv. Confirm song was added to the library
        1. return True if added
        2. False otherwise
f. Define a method retrieve_song_details(self, title: str) -> Union[Dict,str]:
    i. This method uses the title to retrieve the details of a single song
    ii. If song doesn't exist.
        1. return song not found
    iii. Otherwise, return the dictionary with the song details
g. Define a method update_song_details(self, title, artist, genre) -> str:
    i. This method uses the title to allow the user to update the song details.
    ii. If song doesn't exist.
        1. return song not found
    iii. Otherwise, update the song's artist and genre
    iv. Check if the song was updated.
        1. If updated return details updated
        2. Otherwise, return details not updated
h. Define a method delete_song(self, title: str) -> str:
    i. This method uses the title to locate and delete a song.
    ii. If song doesn't exist.
        1. return song not found
    iii. Otherwise, delete the song
    iv. Check if the song was deleted.
        1. If deleted return song deleted
        2. Otherwise return song not deleted
i. Define a method display_all_songs(self) -> Union[str,Dict]:
    i. This method returns all songs in a user's library.
    ii. If the library is empty
        1. Return library is empty
    iii. Otherwise, return the music library dictionary.

4. Define a class UserInput:
    a. This class handles user input validation and returns by interacting with the MusicUser class.
    b. Define a method get_input(self, input_message: str) -> str:
        i. This method gets user input using a custom message.
        ii. While loop
            1. Ask user for input.
            2. If input is empty, tell user input cannot be empty
            3. Otherwise return the user input
    c. Define a method handle_add_user(self) -> object:
        i. This method uses MusicUser object and input to create a user.
        ii. Use the get_input method to get, first and last name.
        iii. Use MusicUser object to create the user
    d. Define a method handle_change_user(self, user_object: MusicUser, username: str) -> MusicUser:
        i. This method uses MusicUser object and username to select a different user.
        ii. Call change_user method from the MusicUser object with username to select that user.
    e. Define a method handle_add_song(self, user_object: MusicUser) -> str:
        i. While loop
            1. This method uses MusicUser object and user input to add a song.
            2. Use the get_input method to get title, artist and genre
            3. Call the add_music_to_library method from the Musicuser object with the inputs to add song to library.
            4. If the song already in the library
                a. Tell the user the song exists so they try again
            5. If the song could not be added, tell the user.
            6. Otherwise, return song added confirmation.
    f. Define a method handle_retrieve_song_details(self, user_object: MusicUser) -> Dict:
        i. This method uses MusicUser object and user input to retrieve
        ii. While loop.
            1. This method uses MusicUser object and user input to retrieve song details.
            2. Use the get_input method to get title.

```
                3. Call the retrieve_song_details method from the Musicuser
                   object with the input get the song details.
                4. If the song was not found
                      a. Tell the user so they will try again
                5. Otherwise, return the song details
    g. Define a method handle_update_song(self, user_object: MusicUser) -> str:
          i. This method uses the MusicUser object and title to select a song for
             updating
                1. While loop.
                      a. Use the get_input method to get title.
                      b. If the song is not in the library
                            i. Tell the user so they will try again
                      c. Otherwise get the user input for artist and genre.
                      d. Use the inputs to update the artist and genre of the song
                         in the library.
                      e. If the song was not updated
                            i. Tell the user so they will try again
                      f. Otherwise return update confirmation.
    h. Define a method handle_delete_song(self, user_object: MusicUser) -> str:
          i. This method uses MusicUser object and user input to delete a song.
         ii. While loop
                1. Use the get_input method to get title.
                2. If the song is not in the library.
                      a. Tell the user so they will try again.
                3. Otherwise, delete the song.
                4. If the song was not deleted
                      a. Tell the user so they will try again
                5. Otherwise, deletion confirmation.
    i. Define a method handle_display_all_songs(self, user_object: MusicUser) ->
       Union[Dict,str]:
          i. This method takes no input and returns all songs in the user library.
         ii. Returns the users music collection.
                1. A dictionary if there are songs
                2. A collection empty message if it is empty.
5. Define a class UserMenu:
    a. The constructor takes no input and initializes the following.
          i. A list of main menu options.
         ii. A list of submenu options.
```

b. Define a private method __user_exists(self, user_object: Optional[User] = None) -> bool:
  i. This method uses the user object to check if at least one user is in the library.
  ii. If the user object is None
    1. Return false
  iii. If we have at least one user
    1. Return true

c. Define a private method def __multiple_users_exist(self, user_object: Optional[User] = None) -> bool:
  i. This method checks if there is more than 1 user.
  ii. Call get_users method from the user_object to return all users.
  iii. If we have more than one user
    1. Return true
  iv. If user object is none
    1. Return false

d. Define a private method def __create_temp_menu(self, list_of_options: List) -> List:
  i. This method dynamically assigns numbers to the menu options that have been dynamically selected.
  ii. Create an empty list to hold temp menu items.
  iii. For loop.
    1. Loop through the list of menu options with enumerate.
    2. Add 1 to the index
    3. Create a string with the index number and menu item.
    4. Append new string with number and menu item to empty tmp list.
    5. Return the temp list

e. Define a method def display_menu(self, user_object: Optional[User] = None) -> List[str]:
  i. This method selects the menu options to display based on user and music state.
  ii. If user doesn't exist
    1. Create a list of menu items based on that state
    2. Call the create temp menu method to add numbers to the filtered list of menu items.
    3. Return add user and exit
  iii. If multiple users exist and we do not have any songs in library

1. Create a list of menu items based on that state
2. Call the create temp menu method to add numbers to the filtered list of menu items.
3. Return add user, change user, add song and exit.

iv. If a single user exists and no songs in library
1. Create a list of menu items based on that state
2. Call the create temp menu method to add numbers to the filtered list of menu items.
3. Return add user, add song and exit.

v. If multiple users exist and we have songs in library
1. Copy the full menu list.
2. Call the create temp menu method to add numbers to the list of menu items.
3. Return the full menu.

vi. If one user exists and the library has songs.
1. Copy the full menu list.
2. Remove the change user option from the list.
3. Call the create temp menu method to add numbers to the list of menu items.
4. Return the temp list.

f. Define a method def display_sub_menu(self, user_object: User) -> List[str]:
   i. Displays the chnage user submenu
   ii. If we have more than one user
      1. Copy the sub menu list
      2. Call the create temp menu method to add numbers to the filtered list of menu items.
      3. Return the change user menu options, add user select user
   iii. Otherwise, tell the user no other users exist and give them an option to add a user.

g. Define a method def display_user_selection_menu(self, user_object: Optional[User] = None) -> List[str]:
   i. This method displays the sub menu based on user state and main menu selection.
   ii. Create an empty list to hold list of users that will be displayed
   iii. Instantiate a counter variable set to zero.
   iv. For loop
      1. Loop through the dictionary getting username and user object.
      2. Increment counter by 1

            3. Append the username to the empty list
            4. Copy the list
            5. Call the create temp menu method to add number options to the list of users.
            6. Return the temp list of users.
6. Define a class Main:
    a. The constructor takes no input and initializes the following.
        i. UserMenu object
        ii. UserInput object
        iii. User counter set to zero
        iv. Current user set to none to hold the active user state and object.
        v. Current username to hold the active user state
    b. Define a method get_menu_number_input(self,input_message: str, num_of_menu_options: int) -> str:
        i. This method gets the number input for the main menu using a custom message and the total number of the menu options.
        ii. While loop.
            1. Get user input, strip whitespace and make it lowercase.
            2. If input is empty
                a. Tell the user the input cannot be empty and tell them try again.
            3. If the input is between 1 and the number of menu options.
                a. Return the user input
            4. Otherwise, tell the user to try again.
    c. Defien a method format_menu_display(self,menu_header: str, current_user: str, user_menu_object_method: object, username_header: str) -> Tuple[List,int]:
        i. This method formats and dynamically displays the correct menu based on state and collects the user's menu selection input.
        ii. Get the list of menu items to display based on the current user state
        iii. Get the length of the list
        iv. Print menu header
        v. If current user is not none
            1. Print the username header
        vi. Print the list of numbered menu options.
        vii. Get the menu number the user selected
        viii. Subtract one from the selected menu number to get the index for the item in the menu list.

```
            ix. Return the menu items and the list index
    d. Define a method print_lightsaber(self) -> None:
            i. Thus method prints a lightsaber as an easter egg.
            ii. Blade as a string
            iii. Hilt as a list
            iv. For loop
                1. Loop through range 6 make the blade print 6 times for length.
            v. For loop
                1. Loop through the hilt list
                    a. Printing each line for the hilt
    e. main(self):
            i. This method runs the program
            ii. While loop
                1. Set menu header string variable
                2. Set username header variable
                3. Format and display main menu and username based on user
                   state and get the input.
                    a. Return the menu and list index
                4. If they select add user
                    a. Call user input object with handle add user method to
                       create user
                    b. Increment the user counter by 1
                    c. If there is no current user
                        i. Update current user and username with the
                           newly created user.
                5. If they select change user.
                    a. Format and display sub menu and username based on
                       user state and get the input.
                    b. If they select add user
                        i. Call user input object with handle add user
                           method to create user
                        ii. Increment the user counter by 1
                        iii. If there is no current user
                            1. Update current user and username with
                               the newly created user.
                    c. If they choose select a user.
                        i. Format and display the select user menu and
                           username based on user state and get the input.
```

        ii. For loop
            1. Loop through list and get selected user.
            2. If the user is found
                a. Strip the display number and add set the username to the username variable.
                b. Call user input object with the handle change user method to update the current user variable with the selected user.
                c. Then update current username with the selected user as well.

6. If they select add a song
    a. Call the user input object with the handle add song method and pass in the current user to add song to their library.
    b. Print confirmation message.
    c. If the select Retrieve song details:
        i. Call the user input object with the handle retrieve song details method and pass in the current user.
        ii. Display the returned song details.
7. If they select update song details
    a. Call the user input object with the handle update song method to update the song.
    b. Print confirmation message.
8. If they select delete song
    a. Call the user input object with the handle delete song method.
    b. Print confirmation message.
9. If they select display all songs.
    a. Call the user input object with the handle display all songs.
    b. If a dictionary is returned
        i. Format and display the song details
    c. Otherwise, print the returned confirmation library is empty.
10. If the user select Exit

        a. Print easter egg.

7. Instantiate the Main class and pass in the Userinput and UserMenu class.
8. Call the instantiated class with the main() method to run the program.