CPSC 501 - Assignment 1

Alex Stevenson - 30073617

A python application to handle downloading the audio from a list of youtube videos as MP3 files. Additional functionality to set MP3 metadata and rename those files once they have been downloaded.

This project uses the `pytest` unit testing framework.

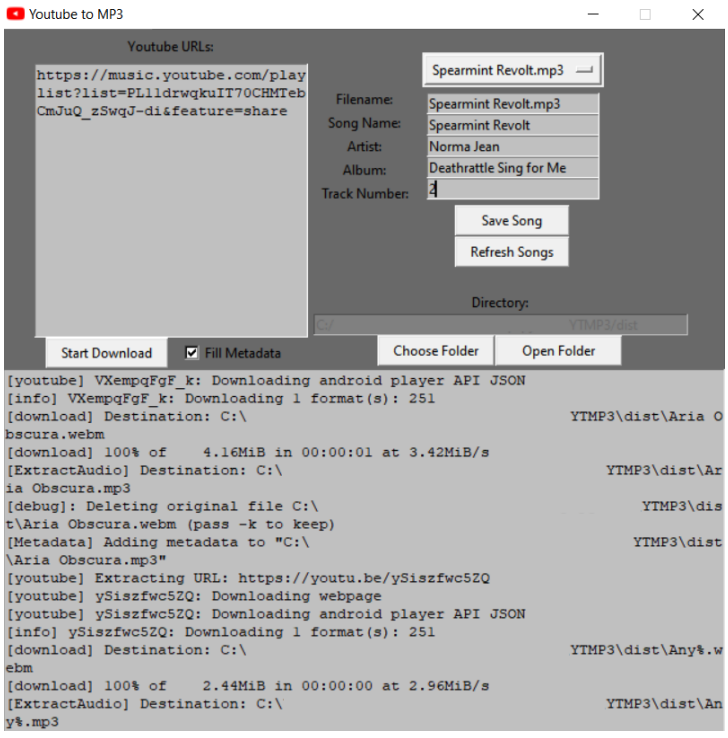
Libraries in use:

- `tkinter`: User interface
- `yt-dlp`: Library that downloads video links from Youtube as MP3 files
- `eyed3`: Library to handle modifying metadata in downloaded MP3s

Github Repository: https://github.com/alexs2112/CPSC501-YTMP3

Initial commit before refactorings are performed:
6edefd8816490cf08f2bf5d50997b79238380211

```
Sep 23, 2023: Add unit tests for metadata
6edefd8816490cf08f2bf5d50997b79238380211
```



Refactoring 1

Git Commit: 3c953ce373c4d2050c2d8e21aba573037f59861f
Improvement: Large Class (*Extract Class*, *Move Method*)

- Basically everything is being handled in the `Application` class of `YTMP3.py`. The main thing to be refactored is the actual download of music files that uses the `yt-dlp` library.
  Steps:
- Performed the `Extract Class` refactoring on every method that was strictly related to downloading files.
- Iteratively touched on methods using the `yt-dlp` library and then followed up on how these changes affected other methods. Used `Move Method` to move those methods into the new class. The class that was extracted is now under `downloader.py`
  Code Snippets:
- Previously, there were several methods in `Application` that are called by `Application.download`. These have all been moved to a new `Downloader` object that is created when the `Application` is instantiated.

```
class Application:
    def __init__(self):
        ...
        self.downloader = Downloader()
        self.window.iconbitmap(self.downloader.executable_path("icon.ico"))
        self.downloader.check_for_ffmpeg(self)
    ...
    def download(self):
        self.downloader.download(
            self,
            self.song_input.get("1.0", tkinter.END).split('\n'),
            self.directory.get(),
            self.metadata.get())
    ...

class Downloader:
    def executable_path(self, path):
        ...
    def check_for_ffmpeg(self, application):
        ...
    def sort_songs(self, directory, songs):
        ...
    def get_playlist_songs(self, application, url):
        ...
    def get_downloader(self, application, directory, embed_metadata):
        ...
    def download(self, application, data, directory, add_metadata):
        ...
```

Testing:

- This change was tested manually, by considering a variety of download links and potential errors while making sure that everything downloaded correctly and messages were displayed as expected. The executable was then built through the `build.bat` script, and the tests were manually performed again.
- Unit testing will need to be implemented after further refactoring is done. Currently the download process requires the entire application to be running to handle logging messages. See *Refactoring 3*.

What Comes Next:

- The code is better structured as this refactoring was able to remove over 100 lines of code from the large `Application` class. This makes it more succinct and ensures that download related code is handled within a download class, instead of all in the main class. This will allow further bug fixes and changes regarding downloads to be created easier.
- As mentioned above, unit tests are currently out of scope for this change as the entire `Application` process gets passed between various download related functions as a parameter to handle logging. A new class to specifically handle logging messages that can be stored in `Application` and passed to `Downloader` would be a much better solution (*Refactoring 3*)
- Another refactoring that can be considered is with the `Downloader.download` method. This method is quite long (50 lines of code) and can be broken down further (*Refactoring 2*)

Refactoring 2

Git Commit: 67849005238c4353f24987fa35c136428ddd6888
Improvement: Large Method (*Extract Method*)

- After *Refactoring 1* it is noticeable that the `Downloader.download` method is quite long and does two separate things. This can be broken into two different methods instead.
- This method essentially does 2 things:
  1. Parse the list of links input by the user captured by the `data` variable. This will return a list of links in a standardized format while also parsing playlists of multiple files to download in a single list.
  2. It then fetches the downloader object that uses `yt-dlp`, called here as `self.get_downloader`, which is then used to iterate over the standardized list of links and downloads them in order.

Steps:

- Performed the `Extract Method` refactoring on `Downloader.download`. It is relatively easy to chop the method into two different methods, now to be known as `process_data` and `download`, where `download` calls `process_data` to return a list of links to download.

Code Snippets:

- The `Downloader.download` function before the refactoring was extremely large and handles two distinct processes.

```
def download(self, application, data, directory, add_metadata):
        # Handle processing user input data
        songs = []
        for link in data:
                ... # Handle URL type between playlist and video links
        if (len(songs) == 0):
                application.error("No songs to download")
                return

        # Actually download user input data
        audio = self.get_downloader(application, directory, add_metadata)
        failed = []
        for i in range(len(songs)):
                url = songs[i]
                try:
                        data = audio.extract_info(url)
                        ...
                except Exception as e:
                        print(e)
                        failed.append(i)
                        continue

        if len(failed) > 0:
                application.error(f"\n{str(len(failed))} Failures Detected")
```

```
                for i in failed:
                        application.print(songs[i])
```

- This is refactored into what follows:

```
def process_data(self, application, data):
        songs = []
        # Process each line of data as above before appending them to `songs`
        return songs

def download(self, application, data, directory, add_metadata):
        songs = self.process_data(application, data)
        # Iterate over `songs`, downloading each file as above
```

Testing:

- Manual testing is performed as above on both individual songs and playlists of multiple songs. This change is isolated to the `download` method so no further testing on the rest of the system is required.

What Comes Next:

- As with *Refactoring 1*, the `Application` object is still being passed between different methods in `Downloader` and is now being passed into the additional method created for this refactoring.
- Unit tests for downloading songs and playlists will be created as part of *Refactoring 3* to ensure that this functionality is never in a failing state.

Refactoring 3

Git Commit: 5cb799f5b98be8888b959ff533e7084b76d0c9aa
Improvement: Inappropriate Intimacy (*Extract Class*, *Move Method*)

- While this is not strictly correct, as python doesn't really have private variables, it is bad practice for the new `Downloader` class to be taking the `Application` class as a parameter just to access various logging methods. These logging methods should otherwise be kept private.

Steps:

- Performed *Extract Class* to create a new `Logger` class and *Move Method* on those logging methods from `Application` into `Logger`. This takes the window console as a constructor parameter so must be initialized in `Application` after the GUI is set up.
- Change each method in `Application` to use the new `Logger` object rather than itself when printing messages to the console.
- Pass the new `Logger` object into the `Downloader` as a constructor parameter. Change each method in `Downloader` to use this `Logger` object rather than passing `Application` into each method.
- The `Download` method in `Downloader` still uses `Application` to add songs to the list of completed songs upon successful download. This is changed to instead pass the `Application.add_song` method into `Download` as a parameter. This is not the best solution, however due to the downloads running in a separate thread it is the best one to not interrupt workflow.
  - An alternative solution would be to return the list of successful songs at the end of `Download` and add them all to the list in `Application`, however this means that the list will *not* update itself while downloads are still running.

Code Snippets:

- New `Logger` class extracted from `Application`

```
class Logger:
    def __init__(self, console):
        self.console = console
        self.last_log = None

    def print(self, msg):
        if self.last_log != None:
            msg = "\n" + msg
        self.console.insert(END, msg)
        self.console.see(END)
        self.last_log = msg

    def debug(self, msg):
        msg.strip()
        if "[download]" in msg and ... # String parsing
            last = self.console.index("end-1c linestart")
            self.console.delete(last, END)
        self.print(msg if "[" in msg else f"[debug]: {msg}")
    def warning(self, msg):
        self.print(f"[warning]: {msg}")
    def error(self, msg):
        self.print(f"[error]: {msg}")
```

- This can then be created in `Application` and passed into `Downloader` so they can both use these logging functions without treading on each others toes.

```
class Application:
        def __init__(self):
                ...
                self.logger = Logger(self.console)
                self.downloader = Downloader(self.logger)
                ...

class Downloader:
    def __init__(self, logger):
        self.logger = logger
```

- Either class can now log messages to the same console by calling

```
self.logger.print("Message")
self.logger.debug("Message")
self.logger.warning("Message")
self.logger.error("Message")
```

Testing:

- Testing the new logging messages through the same manual tests as in *Refactoring 1*
- Now that this `Logger` object is created, unit tests for downloads can be created. Example:

```
def test_download(self):
        data = ["youtube link"]
        self.downloader.download(data, self.directory, False, self.add_song)
        path = os.path.join(self.directory, "Lofi Chill Beats To Study To.webm")
        assert os.path.exists(path)
```

- These new unit tests take a little while due to the nature of the downloads, however it is the most important functionality of the application.
- All unit tests are rerun to ensure no critical functionality has been broken.

What Comes Next:

- A logging object was desperately needed after the previous refactoring. Passing the entire `Application` object into `Downloader` to access the console log was extremely bad practice.
- This refactoring on its own does not lead to any others. The program should have had a logging object from the beginning.

Refactoring 4

Git Commit: 055be880bf9495d601d641c3f0d8316ec72b0878
Improvement: Long Parameter List (*Introduce Parameter Object*, *Move Method*)

- `Downloader.download` is the most important method in the application and requires several different parameters of different types.

```
def download(self, data, directory, add_metadata, add_song_method):
        songs = self.process_data(data)
        ...
```

- These parameters are of type:
  - `data`: List of user-input strings of youtube URLs to download
  - `directory`: A string that is extracted from a `tkinter.Entry` object
  - `add_metadata`: A boolean value that is extracted from a `tkinter.IntVar()` initialized in `Application`
  - `add_song_method`: A method that is called later in `Downloader.download`, introduced as part of Refactoring 3 along with justification for its existence.

Steps:

- Introduce a new Parameter Object called `DownloadData`. Constructor arguments are most of the parameter list for the original method.
  - Change `data` to `input` to be clearer when calling `DownloadData`.
  - Keep `add_song_method` as part of the parameters for `download` as it is a method instead of data.
- Use this new `DownloadData` object as a parameter to `Download` in both `Application` and `test_download` unit tests
- `data` is a local variable that is used twice in `download`. Break this up into `download_data` and `song_data`

Code Snippets:

- New `DownloadData` class, simply stores those parameters in an easy to access object that keeps them together

```
class DownloadData:
    def __init__(self, input, directory, add_metadata):
        self.input = input
        self.directory = directory
        self.add_metadata = add_metadata
```

- New way to call `Downloader.download` in `Application` keeps the data and the method call separate.

```
def download(self):
        download_data = DownloadData(
            self.song_input.get("1.0", tkinter.END).split('\n'),
            self.directory.get(),
            self.metadata.get())
        self.downloader.download(
            download_data,
            self.add_song)
```

Testing:

- A simple and short manual test to download a single song.
- Rerun unit tests that handle downloads.

What Comes Next:

- This makes the code a little cleaner to read as `download_data` is separate from `Downloader.download`. Especially as the three inputs to `DownloadData` are actually method calls to objects storing primitive data types.
- This does not lead to any more refactoring as it is a rather small change.

Refactoring 5

Git Commit: e7fcf1ca2944f71369a7f010144b1d7d25942480
Improvement: Feature Envy (*Extract Class*, *Extract Method*, *Move Method*, *Move Field*)

- The largest refactoring that this codebase requires.
- `Application` contains an overabundance of fields and methods that are solely related to handling the UI interface through the tkinter library.
- These fields and methods can be moved to a new class and then the required ones can be called by `Application`, instead of storing them all in `Application` and only using a few of them for non-UI purposes.

Steps:

- Extract a new `Interface` class to handle all of the tk GUI code. `Application` now creates an `Interface` object
- This class handles all elements created in `Application.setup`, everything regarding button clicks and widget keybindings, everything regarding the directory the application is running in, and everything regarding how the song is displayed and handled in the UI.
  - *Move Method* and *Move Field* are performed repeatedly here.
- Create several helper classes so that `Application` can access important fields in `Interface` (such as `directory` and `metadata` to pass into `Downloader`)
  - Pass the `Application.start_download` method into `Interface` to access this functionality through a button press.
- Move the ability to sort songs from `Downloader.sort_songs` to `Interface`. It makes more sense here as it handles how songs are displayed in the UI.

Code Snippets:

- Several fields that were previously in the constructor for `Application` can now be moved to the new `Interface` constructor, as `Application` never touched them outside of UI purposes.

```
def __init__(self, songs, start_download_method):
        self.last_artist = ""
        self.last_album = ""
        self.metadata = tkinter.IntVar()
        self.selected_song = tkinter.StringVar()
        ...
```

- Certain helper functions are created in `Interface` to provide a cleaner way for other objects to get certain data from the UI

```
def disable_directory(self):
        self.directory.config(state="disabled")

def enable_directory(self):
        self.directory.config(state="normal")

def get_song_input(self):
        return self.song_input.get("1.0", tkinter.END).split('\n')

def get_directory(self):
        return self.directory.get()

def get_metadata(self):
        return self.metadata.get()
```

- `Application` itself is many times smaller. It creates the new `Interface` in its constructor to be able to fetch data and manipulate certain elements of the UI. In `start_download` it is used to block access to changing the `directory` input field while a download is running.

```
def __init__(self):
        ...
        self.interface = Interface(self.songs, self.start_download)
        self.interface.reset_directory()
        self.interface.initialize_songs()
        ...
def start_download(self, _):
        if self.check_thread():
                ...
                self.interface.disable_directory()
```

Testing:

- An additional unit test was added to `test_download.py` that ensures the application does not crash when the user inputs no URLs to download in the Interface.
- Manual testing was done on most of the UI portion. Further automated testing would require testing the whole pipeline of events (input data into `Interface` -> Click button that calls `Application` to start the download -> `Downloader` performs the download) which stops being a unit test.
    - These other individual portions are unit tested already

What Comes Next:

- The code is many times cleaner now that downloading, logging, and the UI are all handled in their own classes instead of the monolithic `Application`.
- One further refactoring will be done for this assignment as `Interface.setup` is over 100 lines of setting up every individual widget and frame in a single method.
- `Interface` could be refactored into two more sections. One handling the `directory` specific methods, and the other handling `song` specific methods.

Refactoring 6

Git Commit: 5ada1b8781b84cb7caf85864fd739291f494eeb2
Improvement: Long Method (*Extract Method*, *Replace Temp with Query*)

- `Interface.setup` (previously `Application.setup`) is one of the ugliest pieces of code I have written in my entire life. It is over 100 lines of setting up tkinter frames and widgets to organize every UI element in a single behemoth of a method. This makes it both extremely confusing and very difficult to make changes to.
- This method can be broken down into several other methods, each new method handling its own section of the UI defined by a frame.
- This method also defines many variables that are stored in the `Interface` object that will be created, a majority of these variables do not need to be stored in `self` and can instead of stored as instance variables.
- Many frames are extremely similar to each other, being created with two constructor parameters of its `master` (the frame or window that owns it) and `bg=self.colour_background`.
  - This can be extracted into a new method with `Replace Temp with Query`, to not have duplicated code everywhere.
  - Every other widget also includes `bg=self.colour_background`, however these are more unique and difficult to turn into a query. They will remain as is for this refactoring
- The ugliest part of the method handles song metadata. It consists of 5 nearly identical chunks of code that each creates a new variable (such as `song_filename`, `song_album`, etc). This could be broken up into iterating over the list of 5 elements.
  - These methods still need to be accessible. This can be turned into a dictionary of text entry fields stored under their metadata name.

Steps:

- Identify different chunks of `Interface.setup` that can be extracted into their own methods. These are typically centered around certain frames.
- For each of these methods, determine which widgets need to remain assigned to `self` and which can be set as local variables.
- Perform *Replace Temp with Query* on all `tkinter.Frame` calls into its own method with a single parameter of `master`.
- Break up the repeated code handling `song_frame` into iterating over a list of attributes that are stored as a class variable.

Code Snippets:

- A portion of the original `setup` method, continue for over 100 lines and you have an unreadable mass of code.

```
def setup(self):
        self.top_frame = tkinter.Frame(bg=self.colour_background)
        self.top_frame.grid(row=0, column=0)
        self.input_frame = tkinter.Frame(master=self.top_frame, ...
        self.input_frame.grid(row=0, column=0)
        self.input_title = tkinter.Label(master=self.input_frame, ...
        self.input_title.pack()
        self.song_input = tkinter.Text(master=self.input_frame, ...)
        self.song_input.pack()
        self.input_frame_buttons = tkinter.Frame(master=self.input_frame, ...)
        self.input_frame_buttons.pack()
        ...
```

- This has been cleaned up to only 15 lines of code.

```
def setup(self, start_download_method):
        top_frame = tkinter.Frame(bg=self.colour_background)
        top_frame.grid(row=0, column=0)
        self.get_console_frame()

        input_frame = self.get_input_frame(top_frame)
        input_frame_buttons = self.get_input_frame_buttons(input_frame,
                                                           start_download_method)
        input_frame_buttons.pack()
        input_frame.grid(row=0, column=0)

        right_frame = self.get_right_frame(top_frame)
        song_frame = self.get_song_frame(right_frame)
        self.get_song_frame_buttons(song_frame)
        self.bind_tab_functionality()
        self.get_directory_frame(right_frame)
        right_frame.grid(row=0, column=1)
```

- A portion of the `song_frame` part of `setup`. This small chunk of code happens 5 times to handle different song metadata. This is now rewritten to iterate over a class variable.

```
# Old Code, this is repeated 5 times
self.field_filename = tkinter.Label(master=self.field_frame,
                                                                    text="Filename:",
padx=10,

bg=self.colour_background)
self.song_filename = tkinter.Entry(master=self.entry_frame, width=25,
                                                                bg=self.colour_foreground)
self.field_filename.pack()
self.song_filename.pack()
```

- This is now rewritten to utilize the below method:

```
METADATA = ["Filename", "Song Name", "Artist", "Album", "Track Number"]
def get_song_metadata_frames(self, field_frame, entry_frame):
        self.song_data = {}
        for metadata in Interface.METADATA:
                field = tkinter.Label(master=field_frame, text=f"{metadata}:",
                                                        padx=10, bg=self.colour_background)
                entry = tkinter.Entry(master=entry_frame, width=25,
                                                        bg=self.colour_foreground, ...)
                field.pack()
                entry.pack()
                self.song_data[metadata] = entry
```

Testing:

- Reran all unit tests
- Manually downloaded an album, ensured all button functionality worked and that tab autocompletion was still there.

What Comes Next:

- This refactoring cleaned up several things that needed to be fixed with `Interface.setup`
  - Reduced the size of the method from over 100 lines to 15 by extracting several other methods. This has been broken up from 1 method into 11!
  - Vastly reduced the number of instance variables that were defined as part of initializing the UI. There are now only 4 that are required by other parts of the codebase.
  - Broke up 5 different instance variables, instead storing those variables in a single directory that is accessed by other parts of the code.
  - Reduced a lot of repeated code by creating a query method that creates a `tkinter.Frame` object.

git log

```
commit 5ada1b8781b84cb7caf85864fd739291f494eeb2 (HEAD -> master, origin/master, origin/HEAD)
Author: alexs2112 <alexander.stevenson@ucalgary.ca>
Date:   Thu Sep 28 11:50:33 2023 -0600
    Refactoring 6: Clean up monolithic setup method

commit e7fcf1ca2944f71369a7f010144b1d7d25942480
Author: alexs2112 <alexander.stevenson@ucalgary.ca>
Date:   Wed Sep 27 13:31:27 2023 -0600
    Refactoring 5: Extract Interface Class

commit 055be880bf9495d601d641c3f0d8316ec72b0878
Author: alexs2112 <alexander.stevenson@ucalgary.ca>
Date:   Tue Sep 26 14:44:21 2023 -0600
    Refactoring 4: Introduce DownloadData parameter object

commit 5cb799f5b98be8888b959ff533e7084b76d0c9aa
Author: alexs2112 <alexander.stevenson@ucalgary.ca>
Date:   Mon Sep 25 12:49:07 2023 -0600
    Refactoring 3: Extract Logging Class

commit 67849005238c4353f24987fa35c136428ddd6888
Author: alexs2112 <alexander.stevenson@ucalgary.ca>
Date:   Mon Sep 25 12:33:47 2023 -0600
    Refactoring 2: Extract Method from Downloader.download

commit 3c953ce373c4d2050c2d8e21aba573037f59861f
Author: alexs2112 <alexander.stevenson@ucalgary.ca>
Date:   Mon Sep 25 12:17:27 2023 -0600
    Refactoring 1: Extract Downloader Class

commit 6edefd8816490cf08f2bf5d50997b79238380211
Author: alexs2112 <alexander.stevenson@ucalgary.ca>
Date:   Sat Sep 23 12:27:03 2023 -0600
    Add unit tests for metadata
```