COMP 202

Fall 2022

Assignment 4

Due: Monday, December 5^{th}, 11:59 p.m.

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	40 points
Question 2:	60 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, allowing the TAs to provide better feedback and not waste time on administrative details.

To get full marks, you must follow all directions below:

- Make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, a 50% penalty per question will be applied.
- Make sure that your code **runs without errors**. Code with errors will receive a very low mark.
- Write your name and student ID in a comment at the top of all `.py` files you hand in.
- Name your variables appropriately. The purpose of each variable should be obvious from the name.
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing. If the line is getting way too long, then it's best to split it up into two different statements.
- Vertical spacing is also important when writing code. Separate each block of code (also within a function) with an empty line.
- **Up to 30% can be removed for bad indentation of your code, omission of docstrings, and/or poor coding style as discussed in our lectures.**

Hints & tips

- **Start early.** Programming projects always take more time than you estimate!
- Do not wait until the last minute to submit your code. **Submit early and often**—a good rule of thumb is to submit every time you finish writing and testing a function.
- Write your code **incrementally**. Don't try to write everything at once. That never works well. Start off with something small and make sure that it works, then add to it gradually, making sure that it works every step of the way.
- Read these instructions and make sure you understand them thoroughly before you start. Ask questions if anything is unclear!
- Seek help when you get stuck! Check our discussion board first to see if your question has already been asked and answered. Ask your question on the discussion board if it hasn't been asked already. Talk

to a TA during office hours if you are having difficulties with programming. Go to the instructor's office hours if you need extra help with understanding a part of the course material.

  - At the same time, beware not to post anything on the discussion board that might give away any part of your solution—this would constitute plagiarism, and the consequences would be unpleasant for everyone involved. If you cannot think of a way to ask your question without giving away part of your solution, then please drop by our office hours.

- If you come to see us in office hours, please do not ask "Here is my program. What's wrong with it?" We expect you to at least make an effort to start to debug your own code, a skill which you are meant to learn as part of this course. Reading through someone else's code is a difficult process—we just don't have the time to read through and understand even a fraction of everyone's code in detail.

  - However, if you show us the work that you've done to narrow down the problem to a specific section of the code, why you think it doesn't work, and what you've tried in order to fix it, it will be much easier to provide you with the specific help you require and we will be happy to do so.

Revisions

Nov. 26:

- Q1: Noted that the minimum number of coins you can transfer is now 25.
- Q2: Fixed a typo in the example for printing a note (the accidental value should be lowercase).
- Q2: Fixed a typo in the description for `upper_octave`.

Part 1 (0 points): Warm-up

*Do **N**OT submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

In the `bookstore.py` file created in lecture, add a function called `create_bookstore`. The function takes as input a string indicating the name of the bookstore, and a list of strings each containing information about a book. The data related to a book is separated by tabs. The function should create a list of objects of type `Book` and then uses this list and the name to create and return a `Bookstore` object.

Warm-up Question 2 (0 points)

In the same module, add a function called `create_bookstore_from_file`. The function takes as input a filename containing information about all the books in this bookstore. Reading the file line by line, the function creates a list of objects of type `Book`. It then uses this list and the name of the file (without its extension) to create and return a bookstore. To test the function use the file `'City_Lights.txt'`.

Warm-up Question 3 (0 points)

In the `Bookstore` class, add a method called `plot_by_genre`. The method should create a bar plot of all the books in the bookstore by genre. Label the x axis with 'Genres', y axis with 'Number of Books', and title the graph with '<Name of the Bookstore> - Books by genre'. The plot should be saved in a file named '<Name of the Bookstore>_by_genre.png'. Test the method with the bookstore obtained by the previous function.

Part 2

The questions in this part of the assignment will be graded.

The main learning objectives for this assignment are:

- Solidify your understanding of nested lists.
- Create simple dictionaries and perform operations on them.
- Solidify your understanding about working with files.
- Understand when and how to use function objects to reduce code repetition and make functions more flexible.
- Apply what you have learned about object oriented programming: defining classes and instance methods, creating instance and class attributes, creating objects and calling methods on them.

Note that this assignment is designed for you to be practicing what you have learned in our lectures up to and including Lecture 28 (OOP III). For this reason, you are NOT allowed to use anything seen after that lecture or not seen in class at all. You will be heavily penalized if you do so.

For full marks, in addition to the points listed on page 1, make sure to add the appropriate documentation string (docstring) to *all* the functions you write. The docstring must contain the following:

- The type contract of the function.
- A description of what the function is expected to do.
- At least three (3) examples of calls to the function. You are allowed to use *at most one* example per function from this PDF.

Examples

For each question, we provide several **examples** of how your code should behave. All examples are given as if you were to call the functions from the shell.

When you upload your code to codePost, some of these examples will be run automatically to test that your code outputs the same as given in the example. However, **it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your assignment, we will run additional, private tests that will use inputs not seen in the examples. You should make sure that your functions work for all the different possible scenarios. Also, when testing your code, know that mindlessly plugging in various different inputs is not enough—it's not the quantity of tests that matters, it's having tests that cover all of the possible scenarios, and that requires thinking about possible scenarios.

Furthermore, please note that your code files **should not contain any function calls in the main body of the program** (i.e., outside of any functions). Code that contains function calls in the main body **will automatically fail the tests on codePost and thus be heavily penalized**. It is OK to place function calls in the main body of your code for testing purposes, but if you do so, make certain that you remove them before submitting. You can also test your functions by calling them from the shell. Please review what you have learned in our lecture on Modules if you'd like to add code to your modules which executes only when you run your files.

Safe Assumptions

For all questions on this assignment, you can safely assume that the **type** of the inputs (both to the functions and those provided to the program by the user) will always be correct. For example, if a function takes as input a string, you can assume that a string will always be provided to it during testing. The same goes for user input. At times you will be required to do some input validation, but this requirement will always be

clearly stated. Otherwise, your functions should work with any possible input that respect the function's description. For example, if the description says that the function takes as input a positive integer, then it should work with all integers greater than 0. If it mentions an integer, then it should work for *any* integer. Make sure to test your functions for edge cases!

Code Repetition

One of the main principles of software development is DRY: Don't Repeat Yourself. One of the main ways we can avoid repeating ourselves in code is by writing functions, then calling the functions when necessary, instead of repeating the code contained within them. Please pay careful attention in the questions of this assignment to not repeat yourself, and instead call previously-defined functions whenever appropriate. As always, you can also add your own helper functions if need be, with the intention of reducing code repetition as much as possible.

Question 1: Paying It Forward (40 points)

An **Application Programming Interface**, or API for short, is a way for two or more computer programs to communicate with each other. An API is comprised of a formal specification, describing how the communication is done, and an implementation, which implements the communication process in code.

APIs are not designed for regular users of a computer program. They are designed for use by programmers, who want their computer program to be able to interface with another.

A common type of API is called a **web API**. A web API is used by a computer program to retrieve information from a website. Most popular websites have their own web API. For example, YouTube, Facebook, Reddit, and Twitter (while it still exists) all have a web API. These APIs allow a computer program to connect to the website and retrieve data, e.g., tweets from Twitter, or posts from Reddit. The data can then be used in the computer program.

Here are some examples of APIs:

- With the YouTube API, we can search for videos, update playlists, make comments, add captions and many other things just by writing code (basically, almost anything that could be done by users of the website itself).
- NASA provides a set of APIs that allow you to retrieve all kinds of data. One simple API allows a computer program to retrieve NASA's Astronomy Picture of the Day.
- Some weather websites have APIs that allow a computer program to retrieve the current weather for a given city. That is how, in the Virtual World, the weather instruments in the Observatory can give you the current weather for Montreal: because the Virtual World server uses a weather website's API to retrieve that information.

Using an API

It is relatively simple to use a web API. Our code must connect to a website at a special API URL and retrieve data from it.

First, we need to know how to retrieve data from a website in Python. To do this, we will need to install the `requests` module into Thonny, following the usual steps. (You may already have it installed if you have tried out the Virtual World.)

With the `requests` module installed, we can use the following code to retrieve data from a given URL. This example assumes that the website will send the information to us in dictionary form (which is not always the case – it depends on the particular API).

```
result_dict = requests.get(url='url-here').json()
```

The resulting dictionary will contain the data requested from the API and/or other information about the status of the request.

Each API has several **endpoints**. An endpoint is a 'point of entry' into the API. Each different request you can make (e.g., posting a comment on a YouTube video, searching for a video, liking or disliking a video) will have its own endpoint. Each endpoint also has its own URL (just like each different webpage on a website). For example, the URL for the YouTube API endpoint that allows for getting a user's like or dislike of a video is `'https://www.googleapis.com/youtube/v3/videos/getRating'`.

After the URL, we must also include certain data to make our request. To continue with the above example, we would need to specify a video ID for the video that we want to check. We do this by adding a question mark (`'?'`) to the end of the URL, followed by (in this case) `'id=VIDEOID'`, where `VIDEOID` is the video ID (i.e., the part after `'https://www.youtube.com/watch?v='` for a YouTube video URL). So the full URL we would connect to would be `'https://www.googleapis.com/youtube/v3/videos/getRating?id=VIDEOID'`[1].

[1]Note that the YouTube API also requires an additional authentication step before connecting to the endpoint URL. We won't be using the YouTube API, but if you are interested, you can read more about it here.

The COMP202COIN API

In this question, we are going to write some code to interface with a web API. Not just any web API – the **COMP202COIN API**, which was designed specifically for this question.

The COMP202COIN API will let you perform certain operations with respect to your coin balance. So that only you can access your own balance, we introduce a security measure called a **token**, which you can think of as a password to your account.

To obtain your token, DM CoinsBot over Slack and enter the command **@CoinsBot token**. Keep your token secure and do not share nor post it anywhere. **We will not be able to resolve any issues for you that are the result of a lost or stolen token**.

The COMP202COIN API, located at 'https://coinsbot202.herokuapp.com/api/', has the following endpoints. The data required for each endpoint is also listed.

- **balance**
  - email: your McGill email address used to sign up to our Slack workspace
  - token: your coin account token
- **transfer**
  - withdrawal_email: your McGill email address used to sign up to our Slack workspace
  - token: your coin account token
  - deposit_email: the McGill email address of the student to which you want to transfer coins
  - amount: the number of coins you want to transfer to the other student

To construct an API request given an endpoint and the needed data, we write the API URL, followed by the endpoint name and a '?', then each piece of data in the form 'name=value', each separated with an ampersand '&'.

E.g., if I wanted to get my balance and my token was ABC, then I would construct the following URL:

'https://coinsbot202.herokuapp.com/api/balance?email=jonathan.campbell@mcgill.ca&token=ABC'

If we then pass this URL into the requests.get function as described on the previous page, we will obtain a dictionary with a key 'status' and a key 'message'. The former will indicate if our request was successful, and the latter will give us the requested data or other relevant information. For example[2]:

```
>>> API_URL = 'https://coinsbot202.herokuapp.com/api/'
>>> request_url = API_URL + 'balance?email=jonathan.campbell@mcgill.ca&token=ABC'
>>> result = requests.get(url=request_url).json()
>>> result
{'message': 'The token in the API request did not match the token that was sent over Slack.',
 'status': 'error'}
```

Through the code that you will write for this question, specified on the next page, you will be able to connect to these two endpoints with your Python program. What will you do with your newfound power? Will you make deals with other students to amass the most COMP202COIN and obtain a coveted sticker? Or will you generously donate your COMP202COIN to those who may need it more than you? The possibilities are endless!

Please use your judgement if and when you make deals with COMP202COIN. That is, don't do anything silly. Don't lend your brand new Tesla to someone just because they give you 5,000 COMP202COIN. Don't exchange real money for COMP202COIN. But if you want to offer your friend some home-made cookies in exchange for some coins, that's great!

[2]The status is 'error' because I didn't use my real token! You must keep your token secure!

For this question, define the following in a file called `coins.py`.

- a function `dict_to_query` that takes a dictionary as input, and returns a string containing the keys and values of the dictionary with the format `'key=value'`, and ampersands (`'&'`) separating each.

```
>>> dict_to_query({'email': 'jonathan.campbell@mcgill.ca', 'token': 'ABC'})
'email=jonathan.campbell@mcgill.ca&token=ABC'
```

- a class `Account`, with a class attribute `API_URL` equal to `'https://coinsbot202.herokuapp.com/api/'`. The `Account` class will store all relevant data about a user's `COMP202COIN` account and also contain methods that operate on that data. Your class should include the following methods:
  - A constructor that takes an email (string) and token (string) as input. Raise an `AssertionError` if the types of the inputs are incorrect or if the email does not end in `'mcgill.ca'`. Creates the following instance attributes:
    - `email`: set to the input of the same name
    - `token`: set to the input of the same name
    - `balance`: set to `-1`
    - `request_log`: set to an empty list

```
>>> my_acct = Account("jonathan.campbell@mcgill.ca", "ABC")
>>> my_acct.balance
-1
```

  - An `__str__` method that returns a string of the format `'EMAIL has balance BALANCE'` where `EMAIL` and `BALANCE` refer to the appropriate instance attributes.

```
>>> my_acct = Account("jonathan.campbell@mcgill.ca", "ABC")
>>> print(my_acct)
jonathan.campbell@mcgill.ca has balance -1
```

  - An instance method `call_api` that takes an endpoint (string) and request dictionary as explicit inputs. If the types of the inputs are incorrect or the endpoint is not valid, raise an `AssertionError`. The method should add the key `'token'` into the dictionary with value given by the instance attribute of the same name. It should then construct an API request URL and send the request as indicated on the previous page. If the `'status'` in the result dictionary is not `'OK'`, raise an `AssertionError` with the value for the `'message'` key as error message. Otherwise, return the result dictionary.

```
>>> my_acct = Account("jonathan.campbell@mcgill.ca", "ABC")
>>> my_acct.call_api("balance", {'email': my_acct.email})
Traceback (most recent call last):
AssertionError: The token in the API request did not match the token that was sent
over Slack.
```

  - An instance method `retrieve_balance` which takes no explicit inputs. Calls the API to retrieve the balance for the current user email. Updates the `balance` attribute of the current user to the given value (after converting to integer), and returns the integer.

    (Note: This example assumes that the correct token was provided.)

```
>>> my_acct = Account("jonathan.campbell@mcgill.ca", "ABC")
>>> my_acct.retrieve_balance()
1
```

– An instance method `transfer` which takes a positive integer amount of coins and an email (string) for the user to which the coins should be transferred as explicit inputs. Raises an `AssertionError` if the types of the inputs are incorrect, if the given email does not end in `'mcgill.ca'`, if the given email is equal to the current user's email, if the current user's balance is `-1`, if the given amount of coins is not positive or if the given amount of coins is greater than the user's current balance. Calls the API to transfer the given amount to coins from the current user to the specified user. Returns the value for the key `'message'` in the result dictionary.

  Note: The minimum number of coins you can transfer is 25.

```
>>> my_acct = Account("jonathan.campbell@mcgill.ca", "ABC")
>>> my_acct.retrieve_balance()
25
>>> my_acct.transfer(25, "alexa.infelise@mail.mcgill.ca")
'You have transferred 25 coins of your balance of 25 coins to alexa.infelise@mail.
mcgill.ca. Your balance is now 0.'
```

We have seen that the `COMP202COIN` API has two endpoints. However, I have heard rumours of a secret, third endpoint to the API. I don't really know about it myself, and we can't give any marks, coins or any other beneficial effect to a student who does uncover any information about it, if it does indeed exist. I will also be trying to find out more about it, and I'll keep you posted with what I find! I think that–**[Hyperlink Blocked]**

Question 2: Making Melodies (60 points)

We've dealt with images in the Assignment 3. Now it's time to make music!

A music melody is made up of a series of notes. Each note has a pitch (a letter ranging from A to G) and a duration. Each note is also located in a particular octave from 1 to 7. Higher octaves correspond to a higher frequency. For example, C4 has a higher frequency than C3. Finally, notes may also have accidentals (sharps or flats) that make them sound different than regular notes. Music also has silences which are called rests.

To play our melody, we will use a third-party Python library called `musicalbeeps`, which we will need to install using the regular steps in Thonny. Depending on your OS and installed software, you may have to download additional software before you can install the library:

- If you are on Windows, you must download the Microsoft C++ Build Tools. When running the installer, select Workloads – Desktop Development with C++, then for Individual Components, select only Windows SDK and C++ x64/x86 build tools. Wait for the download and installation to complete, then install the library in Thonny.
- If you are on Mac, then it may work to install the library right away. Or, you may need to go to the Terminal (in Applications > Utilities) and type in `xcode-select --install` then press return. Wait for the download and installation to complete, then install the library in Thonny.

Note that the installation of the library in Thonny may take several minutes.

Once installed, you can then import the library and use it as follows:

```
import musicalbeeps
player = musicalbeeps.Player()
player.play_note("C4", 1.0)
```

The `play_note` method takes two objects: a note string (with pitch and octave), and a duration (as float). It then plays the specified note through your speakers. The note string can also optionally have a third character: a `'#'` to specify a sharp note, or a `'b'` to specify a flat note.

We will write code in this question to read in song files and play them out of the computer speakers using the `musicalbeeps` library. We will represent notes using scientific pitch notation. In this notation, each note is represented by a letter and a number for its pitch and octave. For example, C4 is a note with pitch C and octave 4. A rest note has pitch R.

Here is an example of a song file. It contains the melody to the popular 'Happy Birthday to you' tune.

```
Happy Birthday
Patty and Mildred J. Hill
.25 D 4 NATURAL false
.25 D 4 NATURAL false
.5 E 4 NATURAL false
.5 D 4 NATURAL false
.5 G 4 NATURAL false
1 F 4 SHARP false
.25 D 4 NATURAL false
.25 D 4 NATURAL false
.5 E 4 NATURAL false
.5 D 4 NATURAL false
.5 A 4 NATURAL false
1 G 4 NATURAL false
.25 D 4 NATURAL false
.25 D 4 NATURAL false
.5 D 5 NATURAL false
```

```
.5 B 4 NATURAL false
.5 G 4 NATURAL false
.5 F 4 SHARP false
1 E 4 NATURAL false
.25 C 5 NATURAL false
.25 C 5 NATURAL false
.5 B 4 NATURAL false
.5 G 4 NATURAL false
.5 A 4 NATURAL false
1.5 G 4 NATURAL false
```

You will observe that the first two lines contain the title and author of the song. Each subsequent line represents a single note. Each note line has five values: the duration of the note (as a float), the pitch, the octave, the accidental value (SHARP, FLAT or NATURAL) and a boolean at the end.

The boolean at the end is known as the repeat value. If a sequence of notes repeats consecutively in the melody, then the repeated notes are not stored in the file. Instead, the sequence that repeats has the repeat value 'true' for the first and last of the repeated sequence, to indicate that the entire sequence should be repeated. For example, given the following lines:

```
0.5 B 4 NATURAL true
0.5 A 4 NATURAL false
1 G 4 NATURAL true
```

then these three notes form a sequence that should be repeated, because they start and end with a 'true' repeat value. If we were loading these lines into our program (as described further below), we would load in six notes: these three, and then the same three repeated right after.

The only special note is a rest note (pitch R). A rest note will not have a value for the octave or accidental. It will only have a duration, then R, then a repeat value.

We will begin this question by writing a `Note` class in a file called `note.py`. This class will contain all information about a single music note and operations that can be performed on it. It should contain two class attributes: `OCTAVE_MIN` and `OCTAVE_MAX`, set to 1 and 7, respectively. It should also contain the following methods:

- A constructor that takes a duration (positive float), pitch (single letter from A to G, or R), octave (integer from 1 to 7) and accidental value (either sharp, flat or natural, in lowercase). The constructor should create instance attributes for each input value. If any input value is not in the correct format as listed above, then raise an `AssertionError`.

  The octave parameter should have a default value of `1` and accidental should have a default value of `'natural'`.

  ```
  >>> note = Note(2.0, "B", 4, "natural")
  >>> note.pitch
  'B'
  ```

- An `__str__` method that returns a string of the format `'DURATION PITCH OCTAVE ACCIDENTAL'` where each of the four words refer to the appropriate instance attributes.

  ```
  >>> note = Note(2.0, "B", 4, "natural")
  >>> print(note)
  2.0 B 4 natural
  ```

- An instance method `play` that takes one explicit input corresponding to a music player object (e.g., the `player` object given in the earlier example would be passed as argument to the method). The method should construct the note string that the `play_note` method accepts (again, described above), then pass the note string and duration to it so that the note can be played through the speakers.

  If the note is a rest note (pitch R), then the note string should be the single word `'pause'`.

  Note: Examples are not needed in the docstring for this method, since it does not return a value and has no effect that we can test in an example.

We will then create a class `Melody` which will store information about a melody of many notes, and operations to perform on them. Write your class in a file called `melody.py` and implement the following methods:

- A constructor that takes a filename (string) as explicit input. You can assume that the file exists and contains a song in the correct format as described above. The constructor should open the file, create an instance attribute for the title and author, then create an instance attribute called `notes` for a list that contains a sequence of `Note` objects, one for every note in the file (and in the same order as they appear in the file).

  If a sequence of notes repeats in the file, then, as described above, a new `Note` object should be created for each repeated note.

```
>>> happy_birthday = Melody("birthday.txt")
>>> len(happy_birthday.notes)
25
>>> print(happy_birthday.notes[5])
1.0 F 4 sharp

>>> hot_cross_buns = Melody("hotcrossbuns.txt")
>>> len(hot_cross_buns.notes)
17
>>> print(happy_birthday.notes[10])
0.5 A 4 natural
```

  Note that the second example above shows the loading of a music file where some notes have repeat set to true. Therefore, although the music file has only 10 note lines, when adding the repeated notes, we get to 17 notes total. That is because the first three notes of the song are repeated (+3), then two sets of two notes are repeated twice (+2, +2).

- An instance method `play` which takes a music player object as explicit input, and calls the `play` method on each `Note` object of the `notes` instance attribute in order, passing the music player object each time as argument.

  Note: Examples are not needed in the docstring for this method, since it does not return a value and has no effect that we can test in an example.

- An instance method `get_total_duration` which takes no explicit inputs and returns the total duration of the song as a float.

```
>>> happy_birthday = Melody("birthday.txt")
>>> happy_birthday.get_total_duration()
13.0

>>> hot_cross_buns = Melody("hotcrossbuns.txt")
>>> hot_cross_buns.get_total_duration()
8.0
```

- An instance method `lower_octave` which takes no explicit inputs. It reduces the octave of all notes in the song by 1 and returns `True`. However, a note's octave cannot be reduced below 1. If that would happen, then do not lower any octaves and instead return `False`. Make sure to use the appropriate attributes of the `Note` class.

```
>>> happy_birthday = Melody("birthday.txt")
>>> happy_birthday.lower_octave()
True
>>> happy_birthday.notes[5].octave
3
```

- An instance method `upper_octave` which takes no explicit inputs. It increases the octave of all notes in the song by 1 and returns `True`. However, a note's octave cannot be increased past 7. If that would happen, then do not increase any octaves and instead return `False`. Make sure to use the appropriate attributes of the `Note` class.

  Note: The above two methods are very similar in what they do. You should consider using a helper function to reduce code repetition.

```
>>> happy_birthday = Melody("birthday.txt")
>>> happy_birthday.upper_octave()
True
>>> happy_birthday.notes[5].octave
5
```

- An instance method `change_tempo` which takes one positive float as explicit input and returns nothing. It should multiply the duration of each note by the given float.

```
>>> happy_birthday = Melody("birthday.txt")
>>> happy_birthday.change_tempo(0.5)
>>> happy_birthday.get_total_duration()
6.5
```

Once you have written both classes, you may run the file `play_melody.py` provided to you with this PDF in order to play one of the song files also provided with this PDF. Or, you can make your own song file! An original, or maybe a version of an existing song you like!

If you do make your own song, please feel free to share it on the discussion board and/or when submitting to codePost! :)

What To Submit

You must submit all your files on codePost (https://codepost.io/). The files you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`coins.py`
`note.py`
`melody.py`
`my_song.txt` (optional)
`README.txt` In this file, you can tell the TA about any issues you ran into while doing this assignment.
Remember that this assignment like all others is an `individual` assignment and must represent the entirety of your own work. You are permitted to verbally discuss it with your peers, as long as no written notes are taken. If you do discuss it with anyone, please make note of those people in this `README.txt` file. If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.

You may make as many submissions as you like, but we will only grade your final submission (all prior ones are automatically deleted).

Note: If you are having trouble, make sure the names of your files are exactly as written above.

Assignment debriefing

In the week(s) following the due date for this assignment, you may be asked to meet with a TA for a 20-25 minute meeting (exact duration TBD). In this meeting, the TA will discuss with you what you should improve for future assignments.

Only your code will determine your grade. You will not be able to provide any clarifications or extra information in order to improve your grade. However, you will have the opportunity to ask for clarifications regarding your grade.

You may also be asked during the meeting to explain portions of your code. Answers to these questions will not be used to determine your grade, but inability to explain your code may be used as evidence to support a charge of plagiarism later in the term.

Details on how to schedule a meeting with the TA will be shared with you in the days following the due date of the assignment.

If you do not attend a meeting, you will receive a 0 for your assignment.