information about the book I downloaded

- description: title: book1: The art of music, Volumn hree: Modern Music; book2: The romantic composers; author of both books: Mason, Daniel Gregory
- url: https://www.gutenberg.org/ebooks/68990, https://www.gutenberg.org/ebooks/58511
- copyright status: book1: public domain in the USA, EBook-No: 68990; book2: public domain in the USA, EBook-No: 58511
- encoding: UTF-8

a question I'd like answered about the books that might require some programming

- are the pronouns in The Art of Music and the romantic composers predominantly male or female?

In [ ]:
```
from collections import Counter


path_list = ['./data/The_Art_of_Music.txt', './data/The_Romantic_Composers.txt']
def count_pronouns(path):
    pronoun_dict = {}
    for i, path in enumerate(path_list):
        with open (path, 'r') as f:
            s = f.read()
        vocab_list = s.lower().split(' ')
        vocab_count = Counter(vocab_list)
        male_pronoun_list = ['he',  'his', 'himself']
        female_pronoun_list = ['she', 'her', 'herself']
        male_count, female_count = 0, 0
        for pronoun in male_pronoun_list:
            try:
                male_count += vocab_count[pronoun]
            except:
                raise KeyError
        for pronoun in female_pronoun_list:
            try:
                female_count += vocab_count[pronoun]
            except:
                raise KeyError
        print(f'Total male pronouns in the book{i+1}:{male_count}, total female pronoun in this book{i+1}: {female
        pronoun_dict[f'book{i+1}']= (male_count, female_count)
    return pronoun_dict

pronoun_dict = count_pronouns(path_list)
print(pronoun_dict)
print('The pronouns in both books are predominantly male.')
```

```
Total male pronouns in the book1:3744, total female pronoun in this book1: 117
Total male pronouns in the book2:1512, total female pronoun in this book2: 67
{'book1': (3744, 117), 'book2': (1512, 67)}
The pronouns in both books are predominantly male.
```

Analysis step:

- read the file as a string and tokenize the string into a list of vocabulary
- construct a counter object called vocab_count to count for some common pronoun words in the list
- compare the count the female pronouns with the count of male pronouns in the list
- conclude that the pronouns in both books are predominantly male.

In [ ]:
```
# Part 1 goes here!
# create two exception class
class DecodeError(Exception):
    pass

class ChunkError(Exception):
    pass

class BitList:
    bitstring = ''

    def __init__(self, binary_str):
        p = set(binary_str)
        s = {'0', '1'}
        if not (s==p or p == {'0'} or p =={'1'}):
            raise ValueError('Format is invalid; does not consist of only 0 and 1')
        self.bitstring = binary_str



    def __str__(self):
        return self.bitstring

    def __eq__(self, other):
        return self.bitstring == other.bitstring

    def arithmetic_shift_left(self):
        self.bitstring = self.bitstring[1:]+'0'

    def arithmetic_shift_right(self):
        self.bitstring = self.bitstring[0] + self.bitstring[:-1]

    def bitwise_and(self, other):
        self_bitstring = self.bitstring
        other_bitstring = other.bitstring
        try:
            b_and = ''
            for i in range(len(self_bitstring)):
                b_and += str(int(self_bitstring[i]) * int(other_bitstring[i]))
            return BitList(b_and)
        except:
            print('length not equal')


    def chunk(self, chunk_length):
        bit_string = self.bitstring
        if len(bit_string)%chunk_length ==0:
            bitlist = []
            for i in range(0, len(bit_string), chunk_length):
                add_chunk = list(bit_string[i:i+chunk_length])
                add_chunk = [int(i) for i in add_chunk]
                bitlist.append(add_chunk)
            return bitlist
        else:
            raise ChunkError()


    def decode(self, encoding='utf-8'):
        if encoding not in ('utf-8', 'us-ascii'):
            raise ValueError('The encoding is not supported.')
        bit_string = self.bitstring
        #print(f'what is bit_string{bit_string}')
        if encoding == 'utf-8':
            if len(bit_string)==8 and bit_string[0]!='0':
                raise DecodeError()

            chunk_list = []
            for i in range(0, len(bit_string), 8):
                chunk_list.append(bit_string[i:i+8])
            #print(f'what is chunk_list:{chunk_list}')

            more_chunk = []
            j = 0
            while j < len(chunk_list):
                chunk = chunk_list[j]
                leading1 = chunk.find('0')
                if leading1>1:
                    small_chunk = chunk_list[j:j+leading1]
                    j += leading1
                elif leading1 ==0:
                    small_chunk = [chunk_list[j]]
                    j +=1
                more_chunk.append(small_chunk)

            #print(f'what is more_chunk:{more_chunk}')

            try:
                decoded_string = ''
                for small_chunk in more_chunk:
                    b = bytes(int(i, 2) for i in small_chunk)
                    decoded_character = b.decode(encoding)
                    decoded_string +=decoded_character
            except:
                raise DecodeError
        else:
            chunk_list = []
            if len(bit_string)%7 !=0:
                raise DecodeError
            else:
                for i in range(0, len(bit_string), 7):
                    chunk_list.append(bit_string[i:i+7])
                try:
                    b = bytes(int(i, 2) for i in chunk_list)
                    decoded_string = b.decode(encoding)
                except:
                    raise DecodeError
        return decoded_string



    @staticmethod
    def from_ints(*args):
        bitstring = ''
        for arg in args:
            bitstring += str(arg)
        p = set(bitstring)
        s = {'0', '1'}
        if not (s==p or p == {'0'} or p =={'1'}):
            raise ValueError('Format is invalid; does not consist of only 0 and 1')
        return BitList(bitstring)
```