```
from optparse import OptionParser
from collections import Counter
import heapq
import os
import re


class HeapNode:
    """
    Class for creating heap object
    Reference - "https://github.com/bhrigu123/huffman-coding/blob/master/huffman.py"
    """
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

    def __eq__(self, other):
        if other is None:
            return False
        if not isinstance(other, HeapNode):
            return False
        return self.freq == other.freq

    def __repr__(self):
        return "Char is {0} and Freq is {1}".format(self.char, self.freq)


class HuffmanCoding:

    def __init__(self, txtfile):
        with open(txtfile, "r") as f:
            self.book_text = f.read()
            f.close()

        self.heap = []
        self.codes = {}

    def text_clean(self, str):
        """
        defunct
        return clean text
        Steps-
        1. lower-casing string
        2. removing all non alphanumeric characters
        :param str:
        :return:
        """
        s = str.lower()
        s = re.sub(r'[^a-zA-Z0-9_]', '', s)
        return s

    def calculate_frequency(self, data_str):
```

```
        """
        Data Cleaning-
        1. removing white spaces
        2. removing '\n' character
        3. lower-casing every character
        4. removing anything that is not a character or a number
        5. removing 0 length characters
        :param data_str:
        :return:dictionary containing frequency of character in the given text
        """
        data_str_sent = data_str.split('\n')
        data_str_token = [s for sent in data_str_sent for s in sent.split(' ')
                          if len(self.text_clean(s)) > 0]
        data_str_char = [s for w in data_str_token for s in list(w)]
        freq_dict = dict(Counter(data_str_char))
        return freq_dict

    def make_heap(self, freq_dict):
        """
        returns heap created out of frequency dictionary
        :param freq_dict:
        :return:
        """
        for key in freq_dict:
            node = HeapNode(key, freq_dict[key])
            heapq.heappush(self.heap, node)

    def merge_nodes(self):
        while len(self.heap) > 1:
            node1 = heapq.heappop(self.heap)
            node2 = heapq.heappop(self.heap)

            merged = HeapNode(None, node1.freq + node2.freq)
            merged.left = node1
            merged.right = node2

            heapq.heappush(self.heap, merged)

    def char_codes(self, node, bit):
        if node.left is None and node.right is None:
            print(node.char, bit)
            self.codes[node.char] = len(bit)

        else:
            self.char_codes(node.left, bit + '0')
            self.char_codes(node.right, bit + '1')


if __name__ == "__main__":
    parser = OptionParser()
    parser.add_option("-f", "--filename", dest="filename")
    (options, args) = parser.parse_args()

    # Initializing the class Huffman Coding
    huffman_coding = HuffmanCoding(options.filename)

    # Reading book
```

```
txt = huffman_coding.book_text
len_txt = len(txt)

# calculating frequency of characters
freq_dict = huffman_coding.calculate_frequency(txt)

# Constructing heap from frequency dict
huffman_coding.make_heap(freq_dict)

# Merging nodes to build heap
huffman_coding.merge_nodes()

# Assigning codes to characters
heap_root = heapq.heappop(huffman_coding.heap)
print('Character codes:')
huffman_coding.char_codes(heap_root, '')
print('Number of characters which are encoded:', len(huffman_coding.codes))
huffman_code_len = {}

# Removing characters whose ASCII values are not between 31 and 128
for c in huffman_coding.codes:
    if 31 < int(ord(c)) < 128:
        huffman_code_len[c] = huffman_coding.codes[c]

print('Number of characters which are encoded between ASCII values 31 and 128:', len(huffman_cc

print('Character Frequency:', huffman_code_len)

# Finding number of bits used for encoding
number_of_bits = 0
for character in huffman_code_len:
    number_of_bits = number_of_bits + huffman_code_len[character] * freq_dict[character]

print("The text was encoded using", number_of_bits, "bits")
print("The text had", len_txt, "valid characters")
print("Using a 7-bit fixed length encoding, this would have been", len_txt * 7, "bits long")
print("So we saved", (len_txt * 7) - number_of_bits, "bits!")
```