FIT9133 Assignment #2
Building a Morse Code Decoder with Python Classes
Semester 1 2018

Dr Jojo Wong
Lecturer, Faculty of IT.
Email: Jojo.Wong@monash.edu

April 15, 2018

Revision Status

$Id: FIT9133-Assignment-02.tex, Version 1.0 2018/04/04 17:30 Jojo $

$Id: FIT9133-Assignment-02.tex, Version 1.1 2018/04/15 16:55 Jojo $

Contents

1 Introduction

This assignment is due on **4 May 2018 (Friday) by 5:00pm**. It is worth **10% of the total unit mark**. A penalty of 5% per day will apply for late submission. Refer to the FIT9133 Unit Guide for the policy on extensions or special considerations.

Note that this is **an individual assignment** and **must be your own work**. Please pay attention to Section 4.2 of this document on the university policies for the *Academic Integrity, Plagiarism and Collusion*.

This second assignment consists of **five tasks** and all the tasks should be submitted as **separate Python modules (classes or executable programs)** with supporting documentation on its usage. All the Python files and any supporting documents should be compressed into one single file for submission. (The submission details are given in Section 4.)

Assessment: Task 1 is weighed 30%**; Task 2 to Task 4 carry equal weightage of marks with** 15% **each; and Task 5 is weighted** 25%**.**

2 Morse Code Interpreter

The Morse Code System

As we have seen in the first assignment, the current International Morse Code system encodes a range of characters, including the ISO basic Latin alphabets (A–Z), some additional Latin letter representing accents, the Arabic numerals (0–9), and a small set of punctuations and procedural signals (known as prosigns).

Each character is presented in Morse Code as an unique sequence of 'dot' (`.`) and 'dashes' (`_`). Figure 1 presents a subset of the characters represented in Morse Code.

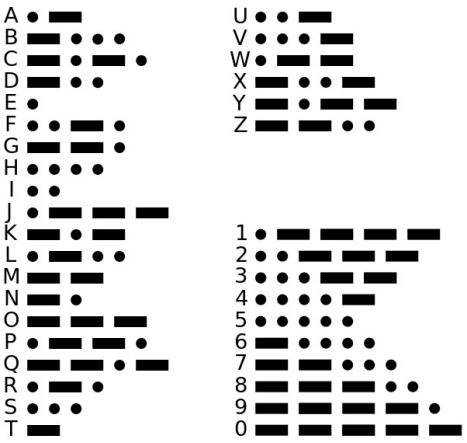

Figure 1: International Morse Code representation (letters and numerals)
[Source: `https://en.wikipedia.org/wiki/Morse_code`]

Each character represented by the Morse Code system is separated by a single space, and each word (i.e. a combination of multiple characters) is separated by three spaces.

The following is an example that includes a punctuation character (the question mark) at the end of the Morse Code sequence. (Note that the punctuation is cosidered as a single word with a single character.)

```
1 Sentence: WHO AM I ?
2 Morse Sequence: .__ .... ___   ._ __   ..   ..__..
```

How Morse Code is represented in this assignment?

In this assignment, we are going to adopt the same set of representation for the Morse Code used in the first assignment based on the binary digits. The 'dots' are represented by the digit 0 and the 'dashes' are represented by the digit 1. As for the spaces, they are represented by the character '*'.

Note that our representation of Morse Code encodes the standard 26 letters (i.e. '**A**' to '**Z**) and the 10 numerals (i.e. '**0**' to '**9**'), with three additional punctuation characters (i.e. the period '**.**'; the comma '**,**'; and the question mark '**?**').

Table 1 defines the set of Morse Code representation used in this assignment.

Character	Morse Code	Character	Morse Code	Character	Morse Code
A	01	**N**	10	**0**	11111
B	1000	**O**	111	**1**	01111
C	1010	**P**	0110	**2**	00111
D	100	**Q**	1101	**3**	00011
E	0	**R**	010	**4**	00001
F	0010	**S**	000	**5**	00000
G	110	**T**	1	**6**	10000
H	0000	**U**	001	**7**	11000
I	00	**V**	0001	**8**	11100
J	0111	**W**	011	**9**	11110
K	101	**X**	1001	**.**	010101
L	0100	**Y**	1011	**,**	110011
M	11	**Z**	1100	**?**	001100

Table 1: Morse Code representation for this assignment (letters, numerals, and punctuations)

For the same example given earlier, our Morse code sequence should read as below:

```
1 Sentence: WHO AM I ?
2 Morse Sequence: 011*0000*111***01*11***00***001100
```

Also note that, we assume the Morse code sequences to be decoded in this assignment represent proper words and sentences in English. Each Morse code sequence should represent a sentence in English.

2.1 Task 1: Building a class for Morse Code decoder

In the first task, you are required to define a class that serves as the Morse Code decoder. This class should have one instance variable which is a dictionary structure that represents each of the Morse Code characters (presented in Table 1) as a string sequence of binary digits (`0` and `1`). This decoder class is used for decoding any Morse Code sequences.

The implementation of this decoder class should include the following three methods:

- `__init__(self)`:
  This is the constructor that is required for creating decoder instances. You should define and initialise the dictionary structure for the Morse Code representation (i.e. the instance variable) in the constructor.
- `__str__(self)`:
  Re-define this method to present the Morse code representation table (i.e. the dictionary structure) in a readable format. You should return a formatted string in this method.
- `decode(self, morse_code_sequence)`:
  This is the method that performs the decoding process. This method should accept a Morse Code sequence as the argument, and attempt to decode it. The decoded message should be returned as a string. (If you encountered an invalid character while decoding, return an error message instead of the partially decoded message.)

  Note: The Morse Code sequence should terminate with one of the three punctuation characters, i.e. the period '**.**'; the comma '**,**'; and the question mark '**?**'.

You should name this class as "Decoder" and the Python file as `StudentID_decoder.py`.

2.2 Task 2: Building a class for analysing decoded characters

In this task, you are required to define a class for analysing the number of occurrences for each of the letters (i.e. '**A**' to '**Z**) and numerals (i.e. '**0**' to '**9**') from the decoded sequences. This class should have one instance variable which is a dictionary structure that is used for keeping track of the number of occurrences for each of the *letters* and *numerals* decoded by the Morse Code decoder in Task 1.

The implementation of this character analyser class should include the following three methods:

- `__init__(self)`:
  This is the constructor that is required for creating instances of this class. You should

define and initialise the dictionary structure for characters (i.e. the instance variable) in the constructor.

- `__str__(self)`:
  Re-define this method to present the number of occurrences for each of the letters and numerals in a readable format. You should return a formatted string in this method.

- `analyse_characters(self, decoded_sequence)`:
  This is the method that performs the analysis on the decoded sequences at the character level. This method should accept a decoded sequence as the argument, and attempt to count the occurrences for each of the letters and numerals encountered in the given decoded sequence. The counts should be updated in the dictionary structure defined in the constructor.

  Note: You should not consider the punctuation characters in this task as they will be handled in Task 4 below.

You should name this class as "CharacterAnalyser" and the corresponding Python file as `StudentID_character.py`.

2.3 Task 3: Building a class for analysing decoded words

In this task, you are required to define a class for analysing the number of occurrences for each of the English words from the decoded sequences. This class should have one instance variable which is a dictionary structure that is used for keeping track of the number of occurrences for each word decoded by the Morse Code decoder in Task 1.

The implementation of this word analyser class should include the following three methods:

- `__init__(self)`:
  This is the constructor that is required for creating instances of this class. You should define and initialise the dictionary structure for words (i.e. the instance variable) in the constructor.

- `__str__(self)`:
  Re-define this method to present the number of occurrences for each word in a readable format. You should return a formatted string in this method.

- `analyse_words(self, decoded_sequence)`:
  This is the method that performs the analysis on the decoded sequences at the word level. This method should accept a decoded sequence as the argument, and attempt to count the occurrences for each word encountered in the given decoded sequence. The

counts should be updated in the dictionary structure defined in the constructor.

Note: Again, you should not consider the punctuation characters in this task as they will be handled in Task 4 below.

You should name this class as "WordAnalyser" and the Python file as `StudentID_word.py`.

2.4 Task 4: Building a class for analysing decoded sentences

Following Task 2 and Task 3, you are required to define a class in this task for analysing the number of occurrences for each type of English sentences from the decoded sequences. This class should have one instance variable which is a dictionary structure that is used for keeping track of the number of occurrences for each type of sentences decoded by the Morse Code decoder in Task 1.

The types of sentence that we are handling in this assignment include: *clauses* (indicated by the comma), *complete sentences* (indicated by the period), and *questions* (indicated by the question mark).

The implementation of this sentence analyser class should include the following three methods:

- `__init__(self)`:
  This is the constructor that is required for creating instances of this class. You should define and initialise the dictionary structure for sentence types (i.e. the instance variable) in the constructor.
- `__str__(self)`:
  Re-define this method to present the number of occurrences for each sentence type in a readable format. You should return a formatted string in this method.
- `analyse_sentences(self, decoded_sequence)`:
  This is the method that performs the analysis on the decoded sequences at the sentence level. This method should accept a decoded sequence as the argument, and attempt to count the occurrences for each sentence type (which could be a number of clauses and/or a full sentence or a question) encountered in the given decoded sequence. The counts should be updated in the dictionary structure defined in the constructor.

You should name this class as "SentenceAnalyser" and the corresponding Python file as `StudentID_sentence.py`.

2.5 Task 5: Putting all the classes together

In this final task, we will test all the classes defined above. You should define a function called `main()` that will drive the flow of execution of the program. Below is a sequence of tasks that should execute within the `main()` function.

- Create an instance for each of the four classes. You should have a decoder, a character analyser, a word analyser, and a sentence analyser.
- Prompt the user to input any random sequences of Morse Code. The Morse Code sequences can be of any length but with the minimum of one set of three consecutive '*'. (Note that the user should be allowed to input multiple Morse Code sequences not until the user indicates that he/she would like to terminate.)
- Invoke the corresponding method in the decoder to perform the decoding on each of the Morse Code sequences provided by the user. (All the decoded sequences should be displayed on the console.)
- Invoke the corresponding method in the character analyser to determine the total number of occurrences for each of the letters and numerals appeared in all the decoded sequences.
- Repeat the same task for the total number of occurrences for each word and each sentence type encountered in all the decoded sequences.
- Display all the analysis results on the console in a readable format.

Note: Some of the tasks above may be implemented as functions. You should make your own decision with respect to this. You should also consider designing a menu with options allowing the user to select which level of analysis is intended (character, word or sentence).

You should name the Python file of this task as `StudentID_main.py`.

3 Important Notes

3.1 Documentation

Commenting your code is essential as part of the assessment criteria (refer to Section 3.2). You should also include comments at the beginning of your program file, which specify your name, your Student ID, the start date and the last modified date of the program, as well as with a high-level description of the program. In-line comments within the program are also part of the required documentation.

3.2 Marking Criteria

The assessment of this assignment will be based on the following marking criteria. The same marking criteria will be applied on both tasks:

- 60% for working program — functionality;
- 10% for code architecture — algorithms, data types, control structures, and use of libraries;
- 10% for coding style — clear logic, clarity in variable/function/class names, and readability;
- 20% for documentation — program comments and user documentation.

Note: As part of the assessment, you are required to attend an interview conducted by your tutor in the prac class in the following week after the assignment has been submitted. (That means if you missed the interview, your assignment will not be graded.)

4 Submission

There will be NO hard copy submission required for this assignment. You are required to submit your assignment as a `.zip` file named with your Student ID. For example, if your Student ID is `12345678`, you would submit a zipped file named `12345678_A2.zip`. Note that marks will be deducted if this requirement is not strictly complied with.

Your submission is via the assignment submission link on the FIT9133 S1 2018 Moodle site by the deadline specified in Section 1, i.e. **4 May 2018 (Friday) by 5:00pm**.

4.1 Deliverables

Your submission should contain the following documents:

- A completed the assignment cover sheet for online submission available on the FIT9133 S1 2018 Moodle site.
- An user documentation (at least 2 pages but not more than 4 pages) in PDF format with clear and complete instructions on how to run your programs. This should be a well formatted document with headings and sub-headings, and a table of contents should be included. (Note that your programs must at least run on the computers in the University's computer labs. Any submission that does not run accordingly will receive no marks.)
- Electronic copies of ALL your files that are needed to run your programs.

Marks will deducted for any of these requirements that are not strictly complied with.

4.2 Academic Integrity: Plagiarism and Collusion

Plagiarism means to take and use another person's ideas and or manner of expressing them and to pass them off as your own by failing to give appropriate acknowledgement. This includes materials sourced from the Internet, staff, other students, and from published and unpublished works.

Collusion means unauthorised collaboration on assessable work (written, oral, or practical) with other people. This occurs when you present group work as your own or as the work of another person. Collusion may be with another Monash student or with people or students external to the University. This applies to work assessed by Monash or another university.

It is your responsibility to make yourself familiar with the University's policies and procedures in the event of suspected breaches of academic integrity. (Note: Students will be asked to attend an interview should such a situation is detected.)

The University's policies are available at: `http://www.monash.edu/students/academic/policies/academic-integrity`