**4CS001 - Coursework 2**

*Data Structures and Iteration*

**This assignment is worth 20% of the overall module grade**

This coursework consists of 2 tasks: A Python task and a Java task. The Python Task is worth 75% of this assessment and the Java task is worth 25% of the assessment.

**Introduction**:

This coding challenge will assess your knowledge of data structures and iteration. It also builds on many of the topics we have covered over the course of the module.

The marking scheme for this task is on Canvas, make sure you check back on a regular basis as you work through the assessment.

**Python Task**

Your task is to develop a program that encodes/decodes Morse Code.

**Task Overview:**

**Morse code** is a character encoding scheme used in telecommunications. It is named after the inventor of the telegraph Samuel Morse and represents textual characters as combinations of long (dashes) and short (dots) electronic pulses, lights or sounds.

For example, the letter J is coded as .---, ! as -.-.-- and 1 as .−−−−

A **single space is used to separate characters within a word** and **3 spaces between words**. Morse is **case-insensitive**, so capital letters are used as a matter of course.

For example, the message **HELLO WORLD** in Morse code is:

.... . .-.. .-.. --- .-- --- .-. .-.. -..

In addition to letters, digits and punctuation, there are some special service codes, the most notorious of which is the international distress signal SOS, coded as ...−−−... These special codes are treated as single characters, but transmitted as separate words. You can experiment with the conversion of text to Morse code and back on many websites online, e.g. <https://morsedecoder.com/>

**Getting Started:**

Start by downloading the file morse\_code.py from Canvas. Add your name and student number to the top of the file. Read the included documentation.

**Program Requirements:**

You will develop a program capable of encoding and decoding Morse code. In order to do this, you will be required to implement several functions, specified in the template provided.

Once complete, your programs *main()* method should do the following:

1. Prompt users to select a mode (encode or decode).

2. Check if the mode the user entered is valid. If not, continue to prompt the user until a valid mode is selected.

3. Prompt the user for the message they would like to encode/decode.

4. Encode/decode the message as appropriate and print the output.

5. Prompt the user whether they would like to encode/decode another message.

6. Check if the user has entered a valid input (y/n) If not, continue to prompt the user until they enter a valid response. Depending upon the response you should either:

a. End the program if the user selects no.

b. Proceed directly to step 2 if the user says yes.

**General Hints:**

• You can make use of str.split() to generate a list of Morse words and characters by using the spaces between words and characters as a separator.

>>> morse = '.... . .-.. .-.. --- .-- --- .-. .-.. -..'

>>> words = morse.split(' ') # 3 spaces used to separate words

>>> words # List containing 2 words in Morse

['.... . .-.. .-.. ---', '.-- --- .-. .-.. -..']

>>> for word in words:

... characters = word.split(' ') # 1 space between letters

... characters

...

['....', '.', '.-..', '.-..', '---'] ['.--', '---', '.-.', '.-..', '-..']

• You will also find str.join() useful for constructing a string from a list of strings.

>>> characters

['.--', '---', '.-.', '.-..', '-..', '-.-.--']

>>> ' '.join(characters)

'.-- --- .-. .-.. -.. -.-.--'

• You should use a loop to keep the programming running if the user says that would like to encode/decode more messages.

• Your program should handle both uppercase and lowercase inputs. You can use

str.upper() and str.lower() to convert strings to a specific case.

**Example Implementation (user input in red):**

Welcome to Wolmorse

This program encodes and decodes Morse code. Would you like to encode (e) or decode (d): g Invalid Mode

Would you like to encode (e) or decode (d): d

What message would you like to decode: .... . .-.. .-.. --- .-

- --- .-. .-.. -.. HELLO WORLD

Would you like to encode/decode another message? (y/n): t

Would you like to encode/decode another message? (y/n): y

Would you like to encode (e) or decode (d): e

What message would you like to encode: Hello WORld

.... . .-.. .-.. --- .-- --- .-. .-.. -..

Would you like to encode/decode another message? (y/n): n

Thanks for using the program, goodbye!

**Structure and Documentation**

The structure of your code and documentation will be analyzed and assessed.

This will be done using a static analysis tool called **Pylint**. This software checks the code in your program, ensures that it follows Python conventions and that all functions, classes and modules have been documented. You can read more about it here: <https://www.pylint.org/>.

Python has an official style guide named PEP8, which is where most Python conventions/coding standards originate from. Example checks that Pylint carries out to ensure that the PEP8 coding standard is followed include things such as:

• checking line-code's length

• checking if variable names are well-formed (snake case)

• checking if imported modules/functions are used

• checking if variables/function parameters are used

It is a good idea to run these checks on your code at regular intervals and before submitting. There are several free websites that allow you to do this e.g. <http://pep8online.com/>

**Note:** Marks will be deducted for warning and errors detected in your code.

**Implementation Details**

**Step 1 – Implementing the *print\_intro* Function:**

To implement this function, you simply need to print an introduction to the user. You should use the following message:

Welcome to Wolmorse

This program encodes and decodes Morse code.

**Step 2 – Implementing the *get\_input* Function:**

To implement this function, you need to request input from the user to determine the mode of conversion and the message that they would like to encode/decode.

Your function should check if the mode the user entered is valid. If not, it should continue to prompt the user until a valid mode is selected.

The function should return two strings, the mode of conversation and the message. The message should be converted to upper case to avoid potential encoding/decoding issues.

**Example Implementation:**

>>> get\_input()

Would you like to encode (e) or decode (d): g

Invalid Mode

Would you like to encode (e) or decode (d): e

What message would you like to encode: Hello WORld

('e', 'HELLO WORLD')

**Step 3 – Implementing the *encode* Function:**

To implement this function, you need to correctly encode a plain text message as Morse Code. It should take a single parameter, the message to be encoded.

You should use the MORSE\_CODE tuple to convert between plain text/Morse code

Your function should return a single value, a string - the encoded Morse.

**Example Implementation:**

>>> encode("Hello World")

".... . .-.. .-.. --- .-- --- .-. .-.. -.."

**Step 4 – Implementing the *decode* Function:**

To implement this function, you need to correctly decode a message in Morse Code. It should take a single parameter, the message to be decoded.

You should use the MORSE\_CODE tuple to convert between plain text/Morse code

Your function should return a single value, a plain text string.

**Example Implementation:**

>>> decode(".... . .-.. .-.. --- .-- --- .-. .-.. -..") "HELLO WORLD"

**Challenge (Extra Credit):**

**Introduction:**

The Morse Code encoder/decoder that you’ve developed works well, however, one major drawback is that it’s only capable of encoding/decoding a single message at a time. Therefore, your task for this challenge will be to attempt to resolve this issue by implementing file reading/writing capabilities so that multiple messages can be encoded/decoded in one go.

To do this, you will need to implement the process\_lines(), write\_lines(), check\_file\_exists and get\_filename\_input() functions.

The first function process\_lines() should take two parameters, the filename from which messages should be read and the mode of operation (encoding/decoding). It should process the lines in the file one by one, returning a list of encoded/decoded messages.

A file has been added to the assessment page on Canvas, morse\_input.txt, that includes a series of plain text strings, which you can use to test your function.

**Example Implementation:**

>>> process\_lines("morse\_input.txt", "e")

['.... . .-.. .-.. ---', '.... . .-.. .-.. --- .-- --- .-.

.-.. -..', '.-- .... .- - -.-.-- ..--..', '... --- ...',

'----- .---- ..--- ...-- ....- ..... -.... --...

---.. ----.', '- .... . --.- ..- .. -.-. -.- -... .-. --

- .-- -. ..-. --- -..- .--- ..- -- .--. ... --- ...- .

.-. - .... . .-.. .- --.. -.-- -.. --- --. .-.-.-']

The second function check\_file\_exists() should take a single parameter, a filename and return a Boolean value (True/False) depending upon whether the file exists within the current folder. There are several ways that you can achieve this, for example using exception handling logic or through the built-in OS module included with Python.

**Example Implementation:**

>>> check\_file\_exists("morse\_input.txt") True

The third function write\_lines() should take a single parameter, a list of strings. It should write these strings to a file called results.txt, with each item taking up a single line.

The function doesn’t need to return anything.

The fourth function get\_filename\_input() should work similarly to the get\_input() function you developed earlier. It shouldn’t take any parameters, but needs to return three values, the mode ("e" or "d" based on whether the user would like to encode/decode), a message if the user would like to process messages using the console (or None) and a filename if the user would like to process data in a file (or None). You should continue to validate the mode that the user enters as well as the input mechanism (file/console).

In addition, you should call your check\_file\_exists() function to verify filenames. Users should be repeatedly prompted to enter the filename if the file doesn’t exist.

**Example Implementation:**

>>> get\_filename\_input()

Would you like to encode (e) or decode (d): g

Invalid Mode

Would you like to encode (e) or decode (d): e

Would you like to read from a file (f) or the console (c)? f

Enter a filename: something\_silly.txt

Invalid Filename

Enter a filename: morse\_input.txt

('e', None, morse\_input.txt)

>>> get\_filename\_input()

Would you like to encode (e) or decode (d): g

Invalid Mode

Would you like to encode (e) or decode (d): e

Would you like to read from a file (f) or the console (c)? c

What message would you like to encode: Hello WORld

('e', 'HELLO WORLD', None)

You will also need to update your main() function to call the functions you’ve implemented.

**Full Example Implementation (user input in red)**

Welcome to Wolmorse

This program encodes and decodes Morse code. Would you like to encode (e) or decode (d): g Invalid Mode

Would you like to encode (e) or decode (d): d

Would you like to read from a file (f) or the console (c)? c

What message would you like to decode: .... . .-.. .-.. --- .-

- --- .-. .-.. -..

HELLO WORLD

Would you like to encode/decode another message? (y/n): y

Would you like to encode (e) or decode (d): e

Would you like to read from a file (f) or the console (c)? c

What message would you like to encode: Hello WORld

.... . .-.. .-.. --- .-- --- .-. .-.. -..

Would you like to encode/decode another message? (y/n): y

Would you like to encode (e) or decode (d): e

Would you like to read from a file (f) or the console (c)? f

Enter a filename: something\_silly.txt

Invalid Filename

Enter a filename: morse\_input.txt

Output written to results.txt

Would you like to encode/decode another message? (y/n): n

Thanks for using the program, goodbye!

**Submission and Marking**

You should upload your submission to Canvas. If you fail to do so, you will receive a grade of

0 NS (Non-Submission). The deadline for doing so is the 15th November 2020 at 14:00.

Your work will be automatically marked using a program that tests each of the individual functions you have implemented. Therefore, it is very important that you do not alter any of the function signatures in the template and implement everything as instructed.

You will receive an individualised test report, showing which of the tests passed and failed. Spot checks will be carried out to ensure that the marking is both fair and consistent. However, if you feel that you have been unfairly penalised, please do not hesitate to get in touch.

**Java Task**

This assessment will test your knowledge of arrays in Java. You will be given a file called WordSearchSolver.java which will contain a 2D char array (word search grid). You will also be given a list of words (below). Your task is to create a program which finds the location of those words within the grid by printing out the position of the first and last character. Bear in mind that in Java, position 0,0 is located at the top left char of the grid. Also, some of the words inside the list below do not exist within the grid, you need to alert the user which words cannot be found. You should use your problem-solving skills to design and implement an algorithm which finds these words in whichever way you want. Play the game first using the wordsearch.jpg file which has the grid already laid out for you in a more readable format. Whilst you play it, make a note of what strategy your using, this logic should then be applied within your searching algorithm. Here is the list of words:

GRUESOME MARRIED SCATTERED YOUTHFUL FEDERAL

UNITED POISED WONDERFUL DISTINCT ABROAD

IMPORTANT FREQUENT WOLVES POCKETS ADVANCED

Instructions on how to play a word search:

*You are given a grid of jumbled up letters which have words hidden inside them. Your job is to find where those words are, if this was on paper, you would highlight those words using a marker but as we are using an array and java code, you will only need to print out the position of where these words are. The words are placed using different orientations, meaning some are written vertically, horizontally, or diagonally.*

You can use the following set of steps to implement your code:

1. Within the WordSearchSolver.java file, you’ll be given a 2D array within the main. You should create an array of strings to store all the words (above) you’re trying to find.
2. Using a series of loops, pick the first word from the word array. Then loop through all the individual characters within the 2D array and match it with the first letter of the word.
3. Then apply another condition which checks the second letter of the word with the surrounding characters of the first letter. Do this for the remaining length of the word, for example, if the word you’re looking for is WOLVES, you should be checking a total of 6 letters surrounding the initial letter of W. You will then check for O, if it’s there, then check for L etc.
4. If the word is found, print out the coordinates of the first letter and last letter.
5. If the word is not found, print out the word followed by “is not found within this grid.”