Computer Programming from Speech and Language Processing: Assignment 2

1 Introduction - Speech Synthesiser

For this assignment you will be required to create a **Speech Synthesiser** Python program. This will be a very basic waveform concatenation system, whereby the acoustic units are recordings of monophones. You will be provided with several files that you must use to do this:-

SimpleAudio.py
: This is a version of the `SimpleAudio.py` module that we have used in the lab sessions. The `Audio` class will allow you to save, load and play `.wav` files as well as perform some simple audio processing functions. You should **not** modify this file.

synth.py
: This is a skeleton structure of the program. Your task is to fill in the missing components to make it work. You are free to add any classes, methods or functions that you wish but you must **not** change the existing argparse arguments.

monophones/
: A folder containing `.wav` files of monophones.

examples/
: A folder containing example `.wav` files of how the synthesiser should sound.

2 Task 1 - Basic Synthesis

The primary task for this assignment is to design a program that takes an input phrase and synthesises it. The main steps in the procedure are as follows:-

- normalise the text (convert to lower/upper case, remove punctuation, etc.)
- convert the word sequence to a phone sequence – you should make use of `nltk.corpus.cmudict` to do this.
- concatenate the monophone wav files together in the right order to produce synthesised audio.

Your program should work by executing the `synth.py` script from the command line with arguments, e.g. the following should play "hello":-

```
python synth.py -p "hello"
```

If a word is not in the **cmudict** then you should print a warning to the user and exit the program.

You can also listen to the examples `hello.wav` and `rose.wav` which were created as follows:-

```
python synth.py -o hello.wav "hello nice to meet you"
python synth.py -o rose.wav "A rose by any other name would smell as sweet"
```

If you execute the same commands with your program and the output sounds the same then it is likely you have a functioning basic synthesiser.

3 Task 2 - Extending the Functionality

Implement **at least two** of the following extensions:-

Extension A – Volume Control
Allow the user to set the volume argument (`--volume, -v`) to a value between 0.0 and 1.0. You should use the `rescale` method from the `Audio` class to do this.

Extension B – Punctuation
If the input phrase contains a comma – insert 250ms of silence.
If it contains a period, question mark or exclamation mark – insert 500ms of silence.
Strip all other punctuation.

Extension C – Spelling
Allow the user to set the spell argument (`--spell, -s`) that will synthesise spelling instead of pronunciation. Do this by converting a string into a sequence of letters, then to an appropriate phone sequence to pronounce for each letter in its alphabetic form.

Extension D – Text Normalisation for Numbers
If the input phrase contains numbers in numerical form, convert them to word sequences, e.g. –

```
"The meaning of life is 42"
-> "the meaning of life is forty two"
```

You should include the ability to normalise numbers from 0 up to at least 999 (”nine hundred and ninety nine”).
Decimal points can be handled by reciting ”point” and then reading the numbers after the point as individual digits, e.g. –

```
"Pi is about 3.14159"
-> "pi is about three point one four one five nine"
```

Take care not to treat decimal points as periods (see **Punctuation**).

Extension E – Text Normalisation for Dates
If the string contains dates in the form DD/MM, DD/MM/YY or DD/MM/YYYY, then convert them to word sequences, e.g. –

```
"Burns Night is 25/01"
-> "burns night is the twenty fifth of january"

"John Lennon died on 1/12/80"
-> "john lennon died on the first of december nineteen eighty"
```

You may wish to make use of the built-in `datetime` and/or `re` packages to do this.

4 Rules and Assessment

Your submission should abide by the following rules:-

- you are encouraged to discuss the assignment together but all submissions must be written individually and be your **own work**.
- you may only use **numpy**, **nltk**, the provided files, and any packages that are **built-in** to Python.
- you may **not** change any of the existing arguments provided in `synth.py`.

The assignments will be graded out of 100 and will be assessed according to the following criteria:-

Task 1 (40 marks)
Your system:-

- is able to synthesise several test phrases that contain only words (no punctuation, numbers, etc.).
- can handle out of vocabulary words.
- is able to play and/or save the output to a file

Task 2 (40 marks)
You must implement at least 2 of the extensions in order to pass.
However, for a higher mark, you can implement more of the extensions or present particularly strong solutions. For example, 3 extensions implemented to a high standard could achieve the same grade as 4 extensions implemented less comprehensively.

Style and Design (20 marks)
We will also award marks based on how well your code is designed and presented. Take care to use appropriate object-orientated design as well as suitable naming and comments.

You will be required to submit your `synth.py` file at the end.

Submissions are marked anonymously. You **must** prepend your exam number (the 'B' number on your student card) to the file prior to submitting it, e.g. your submitted file should be in the format:

`B123456_synth.py`

In addition, do **not** include any identifying content (e.g. Matriculation number, name etc.) in the submitted file.