CMPSC 102
Discrete Structures
Fall 2019

Practical 7: Using Generator Functions to Output Plain Text from Morse Code

Refer to your notes, slides and sample Python code from this week and other weeks. In particular, follow the python code that we created in class this week: `yay.py` *to help you see how strings are implemented in generators.*

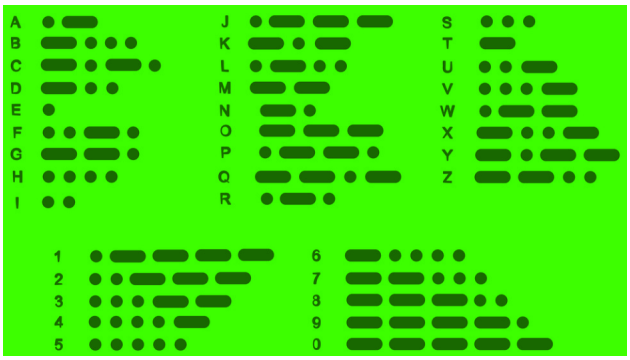

Figure 1: Morse code (written in *dots* and *dashes*) was used to send messages over telegraphs and early radio communication (before the voice could be transmitted). In the latter part of the 19th and early 20th centuries, seemingly all high-speed international communication was encoded in Morse code to be sent through telegraph lines, undersea cables and radio circuits. Take that, Internet!

Summary

A class of functions in Python3 rely on generative iterators to drive their operations of calculation. The iterators do not compute all values at once that must then be held in memory. On the contrary, a generator function, using a `yield` or `next()` line of code, is able to turn-out values only as they are needed. This type of coding reduces the amount of memory required to complete iterative types of tasks where memory consumption is a major consideration.

Another important part of this practical is to work with Morse code which was developed to encode messages before the voice could be transmitted over radio and telephone lines. Ships used to use Morse code to send and receive all their communications to each other and to the shore. The signals were also used to send emergency messages and urgent requests for supplies to surrounding ships. Shown in Figure 2, such an urgent request is being made for coffee.

In this practical, we will be completing Python3 code to incorporate generator functions to work with strings to translate them into Morse code outputs, shown in Figure 1. **Your job is to complete a decoder function to convert Morse code back into text using a generator function and a dictionary.** Although, translating strings from Morse code into plain text could be done in absence of generator functions in this code, there is no amusement in such a silly task.

Handed out: 11^{th} Oct 2019

GitHub Starter Link

https://classroom.github.com/a/XygOyvMO

To use this link, please follow the steps below.

- Click on the link and accept the assignment.
- Once the importing task has completed, click on the created assignment link which will take you to your newly created GitHub repository for this lab.
- Clone this repository (bearing your name) and work on the practical locally.
- As you are working on your practical, you are to commit and push regularly. You can use the following commands to add a single file, you must be in the directory where the file is located (or add the path to the file in the command):
  - `git commit <nameOfFile> -m ‘‘Your notes about commit here’’`
  - `git push`

  Alternatively, you can use the following commands to add multiple files from your repository:
  - `git add -A`
  - `git commit -m ‘‘Your notes about commit here’’`
  - `git push`

What to do for this practical

Locate the incomplete Python3 code from the file `src/myMorseGenerator_starter.py`. This code will already encode text inputs into Morse code using a type of generator function called `myMorseEncoder()`. Here, you will be completing the function, `myMorseDecoder()` to implement another generator that will translate a user-entered Morse code string back into plain text. You are then to play with the program with the same level of amazement and excitement as the two gentlemen featured in Figure 2.

Figure 2: Morse code was used to signal emergency messages from ships to other ships or to shore commands. Here, a captain and operator request help by telegraphing an S-O-S message in Morse code to arrange a rescue party. The signal was sent using a single keypad that was used to prepare the *dots* or *dashes* that make up the code. Note that the emergency message was able to be sent in absence of any WIFI or online service. Take that, Internet! (Drop-mic)

Output

You might want to consider adding a print statement to prove that your `type` is actually a generator, as shown below in the output.

The output is the following for an input of plain text.

```
Input your message (text or encoded) : coffee
  c  :  -.-.
  o  :  ---
  f  :  ..-.
  f  :  ..-.
  e  :  .
  e  :  .
Generator component: encoding
Type : <class 'generator'>
 myMorseEncoder()
Message:  -.-. --- ..-. ..-. . .
```

Your task is to create a decoder, the output of which is following for an input of Morse code.

```
Input your message (text or encoded) : -.-. --- ..-. ..-. . .
Generator component: encoding
Type : <class 'generator'>
 myMorseDecoder()
Message:  COFFEE
```

Questions-In-Blue

Please consider the following questions for your mini-reflection deliverable.

1. Show the output from your encoder and decoder using an example piece of text.
2. How does your code work using generator functions?
3. How could the code be improved in any way to improve its efficiency or usefulness?

Deliverables

1. Your completed (and working) Python3 code (`src/myMorseGenerator_starter_i.py`) with a working `myMorseDecoder` function that applies a generator function to decode user-supplied Morse code into plain text.
2. Mini-reflection document `writing/minireflection.md`

```
--. --- / --. . - / .----. . --
```