layout: page
title: COMP151 - Project 1 - Monoalphabetic Substitution Ciphers
permalink: /teaching/COMP152/projectst/monocipher

mathjax: true

Key Sequence Files

One line. A *permutation* of the alphabet in uppercase. It's the cipher sequence, i.e. the letters you see in cipher text. The first letter in the sequence corresponds to plain text *a*, the next *b*, and so on.

Cipher Text Files

All uppercase letters, in blocks of
five letters separated by spaces, with
five blocks per row.

Plain Text Files

If the file has not be enciphered, then it's just English. This includes
punctuation. If the file is the
result of an decipherment, then it
has the same structure as a cipher text file (5 letter blocks, 5 blocks per row), but letters are lowercase
to indicate that it's plain text, not
cipher text.

Project Lab Assignment - From Key File to Dictionary

Feel free to use any python built-in functions and in particular, try to make good use of list and dictionary comprehension when possible. It is possible to write each of these function without writing any loops.

1. Write and test a function named `buildkey` that takes one string, the name of a file containing a key sequence, and returns a dictionary suitable for decipherment where keys are the cipher letters (what's in the file, uppercase letter) and the associated values are the plain text letters (what's not in the file, the lowercase letters).
2. Write and test a function named `prepareplaintext` that takes a single string, then removes anything that is not a letter and converts all the remaining letters to lowercase. The resultant string is then returned.
3. (Optional) Write and test a function named `blocktext` that takes a string and splits it into a list of 5 letter groups. If the string's length is not a multiple of 5, then the last group can and should be the remaining 1 to 4 letters.

Version 0 - Decipher

```
$ python3 monocipher.py -d cipher.txt key.txt plain.txt
```

Given the key sequence found in the file `key.txt`, decipher the the cipher text found in cipher.text and write the resultant plain text to the file plain.txt. In general, any file names will do, but order matters: cipher, key, then plain.

Version 1 - Encipher

```
$ python3 monocipher.py -e plain.txt key.txt cipher.txt
```

Given the key sequence found in the file `key.txt`, encipher the the plain text found in plain.text and write the resultant cipher text to the file cipher.txt. In general, any file names will do, but order matters: cipher, key, then plain. The cipher text file should conform to the cipher text file format described above.

All version 0 functionality should still work. A complete version 1 program should be capable of both encipherment and decipherment.

Version 2 - Key Generation

```
$ python3 monocipher.py -k key.txt -s 4
```

Generate a cipher sequence by shifting the standard alphabet. Shift amounts can be positive or negative. Positive shifts move letters right where negative move them left. The above example is a right shift of 4. The resultant sequence should be written to the file key.txt

```
$ python3 monocipher.py -k key.txt -r
```

Generate a cipher sequence by reversing the standard sequence. The resultant sequence should be written to the file key.txt.

```
$ python3 monocipher.py -k key.txt -s 4 -r
$ python3 monocipher.py -k key.txt -r -s 4
```

These operations can be combined and done in any order. The order they appear in the command is the order they should be carried out. They cannot, however, be repeated. Nothing like the following is allowed.

```
$ python3 monocipher.py -k key.txt -s 4 -r -s -17 -r
```

The program should maintain the functionality of all previous versions.

Version 3 - Expanded Key Generation

```
$ python3 monocipher.py -k key.txt -kw Douglas
```

Add a third option to key generation, keywords. This option has two parts. The first is the `-kw` for signaling a keyword-based generation, the second is the actual keyword. *If keyword-based generation is used it* **must** *come before shifts or reverses.