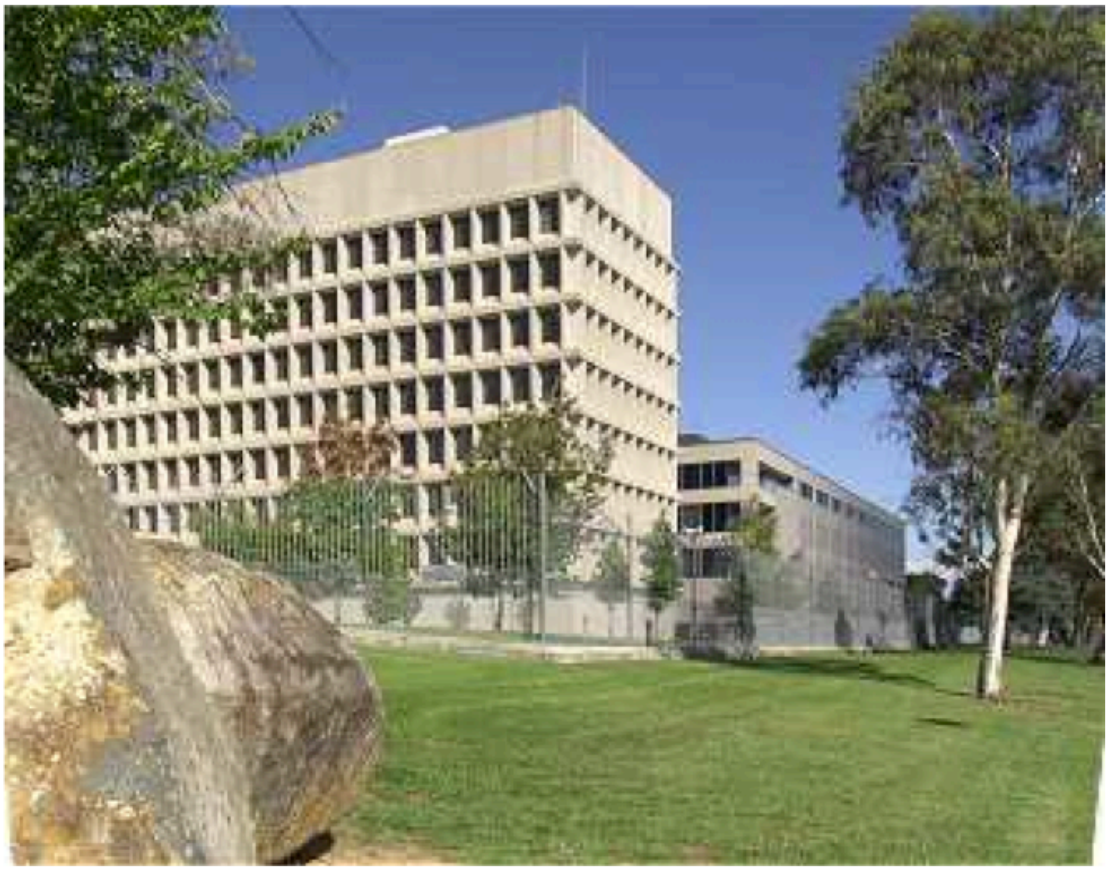
Portfolio 1 Task: 'Crypto Reverso' (easy)

Overview

This task involves developing two Python functions that manipulate lists and strings.

The Defence Signals Directorate (DSD) is an intelligence agency in the Australian Government's Department of Defence. It performs signals intelligence and information security services of national significance, a large part of which inevitably involves cryptography.

DSD's headquarters is in Canberra on the shores of Lake Burley-Griffin.



At the entrance to this building is a small brass plaque bearing a legend something like the following:

```
XECN ELIG IVXL ANRE
TEXS IXMO DEER FXFO
XECI RPXE HTXX
```

Before you read any further, see if you can decipher this mysterious message. The solution is overleaf…

Although it looks like this is a complex encrypted message, it's actually written in disguised plain text. The message was "encrypted" by replacing the spaces between words with an "**X**", reversing the text and splitting the text up into blocks of four characters.

By reading the text backwards, ignoring the layout, and treating **X**s as spaces we can see that the message is

THE PRICE OF FREEDOM IS ETERNAL VIGILENCE.

The layout and spacing are designed to fool you into thinking the code is more complicated than it really is. In fact the size of the "blocks" of characters and their arrangement is not significant at all. Indeed, this is not even a true encryption function because we can decrypt the message without the need for a key; all we need is knowledge of the process used to disguise the message.

In this task we will develop a similar, but stronger, encryption process in which the original key used to encrypt the message is needed to decrypt it successfully.

Our encryption process

For simplicity we assume the messages to be encrypted consist of just upper-case alphabetic characters and spaces. Our encryption process involves the following steps (but not necessarily in the order listed below).

- Spaces in the text are replace by **X**s. (This just makes the encrypted text harder to read. Also we must assume that the message to be encrypted does not already contain any **X**s.)
- Each word in the original message is reversed. For example '`VIGILENCE`' becomes '`ECNELIGIV`'.
- Consecutive sequences of words, called "blocks", have the order of the words reversed. The size of these blocks is a parameter to the encryption function and acts as the "key". For example, if the message was `'THE PRICE OF FREEDOM IS ETERNAL VIGILENCE'` and the block size is four then each block of four words will be reversed, producing `'FREEDOM OF PRICE THE VIGILENCE ETERNAL IS'` (ignoring the other two steps above). Notice that the last block only has three words in this case, but it is still reversed nonetheless.

Putting these three transformations together produces our encryption process. For instance, given the message `'THE PRICE OF FREEDOM IS ETERNAL VIGILENCE'` and a block size of four, the encryption function will return

`'MODEERFXFOXECIRPXEHTXECNELIGIVXLANRETEXSI'`.

(The colours here are just to make the correspondence between the original message and the encrypted text easier to see.) The corresponding decryption process is almost identical except that the **X**s are replaced by spaces. The individual words and blocks of words are reversed in exactly the same way as they were during encryption to recover the original message.

Enclosed with these instructions is an incomplete Python program called `crypto_reverso_Q.py`. This program contains a set of unit tests that you must pass by developing the missing functions `encrypt` and `decrypt`. Study these tests for more examples.

Specific steps

The provided Python program `crypto_reverso_Q.py` contains a set of unit tests that you must pass by developing the missing functions `encrypt` and `decrypt`. These functions must do the following.

- The `encrypt` function must accept a plaintext string and a block size and return the encrypted version of the string, following the process outlined above. For instance, the function call

  ```
  encrypt('PARANOIA IS OUR PROFESSION', 3)
  ```

  should return the string

  ```
  'RUOXSIXAIONARAPXNOISSEFORP'.
  ```

  Your function should work with any length string and any positive "block size".

- The `decrypt` function should reverse this process to reveal the original plaintext message given the ciphertext. For instance, the function call

  ```
  decrypt('TONXSIXYCAVIRPXEVISSAPXEHTXROF', 3)
  ```

  should return the string

  ```
  'PRIVACY IS NOT FOR THE PASSIVE'.
  ```

  Your function should work with any length string and any positive "block size".

Consult the supplied unit tests for more examples of the required behaviour of these two functions.

When we test your program we will use a different set of unit tests, so your code must work in general, not just for the supplied tests.

Development hints

1. The encryption and decryption functions are almost identical in this case. Having completed the first you should have no trouble completing the second.
2. Although they sound complicated, both of these functions can be expressed very consisely in Python. If you find yourself writing a lot of code then you are probably using the wrong solution strategy!
3. Most of the work involved in completing this task requires appropriate manipulation of lists and text strings. You will need to use list slicing and sequence reversal operations just like those covered in the Week 2 lecture and workshops. You should also consult the Python Library Reference for helpful operations that can be applied to sequence types, such as those used to split strings into lists and join list of strings.

Deliverables

The deliverables for this task are as follows.

1. A Python program called `crypto_reverso.py` that implements the encryption and decryption functions described above. This program must work under Python Version 2.7. Create your program from the supplied `crypto_reverso_Q.py` template, with the '`_Q`' suffix removed from the file name.
2. A completed "statement" at the beginning of the Python file to identify your programming pair. (We can't award you marks if we don't know who you are!) Also you must indicate the percentage contribution made to the *whole* portfolio by both students. We do *not* expect a precise 50/50 split; an accurate and honest assessment is appreciated. This percentage will not affect your marks *except* in cases of extreme imbalances or disagreements.

Apart from working correctly, your program code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable and function names.