In our experience, there are two features of the Enigma project that cause the most confusion in students: understanding how the rotors implement each step of the translation and figuring out how to implement both directions of the rotor operation in Python. The problems on this week's section are designed to help you work through each of these areas of conceptual difficulty.

1. **Tracing the Enigma Rotors**

   Figure 1 below shows an image of one of the Enigma rotors from two different perspectives: an edge-on view of the rotor along its axis and the "unrolled" view that shows the connections between the two sides of the rotors. The colored line in the figure shows that, in this position with the offset equal to its initial value of 0, position 0 on the right side of the rotor is connected to position 4 on the left side, which corresponds to mapping **A** to **E**.

   Use the diagram in Figure 1 to answer the following questions:

   (a) With this current offset of 0, where does a signal starting ot position 1 (**B**) on the right go on the left?

   (b) When the rotor advances to offset 1, where does the signal from position 9 (**J**) go?

   (c) When the rotor advances 3 more times to offset 4, where does a signal originating from position 2 (**C**) go?

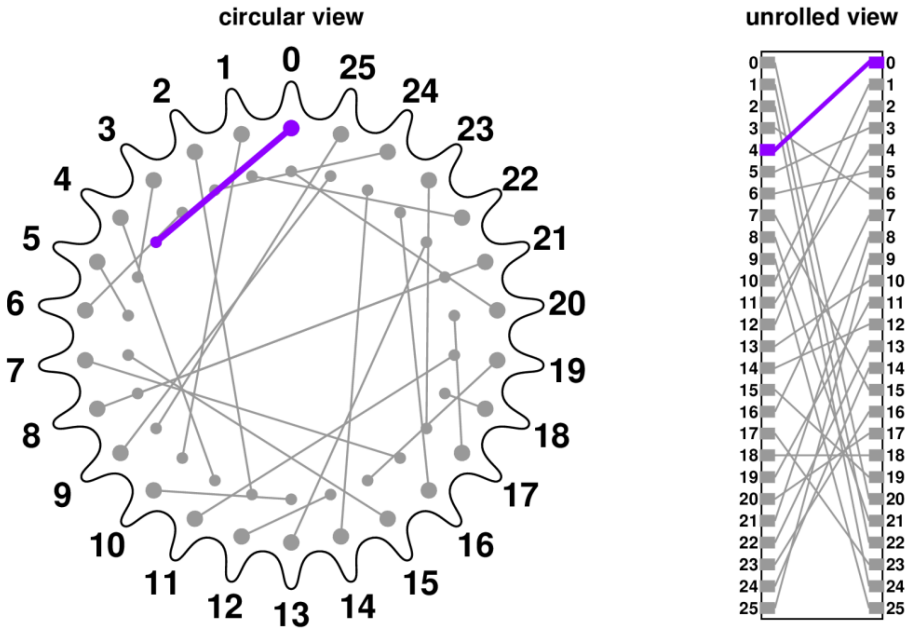

Figure 1: Two views of an Enigma rotor.

2. **Coding the `apply_permutation` method**

   The Enigma project guide suggests that getting the rotor transformations to work is a bit easier if you implement a top-level `apply_permutation` function that implements the following pseudocode:

```
def apply_permutation(index, permutation, offset):
    Compute a new index by shifting the index by the offset, wrapping if needed
    Use than new index to look up the corresponding letter in the permutation string
    Convert that letter to a number corresponding to its location in the alphabet
    Shift that number back by the offset, wrapping as necessary
```

   Your task in this problem is to transform this pseudocode into Python and then to write a small unit test to ensure that it works. Once you have this working, it is a relatively small step to implement it into your `EnigmaModel` class for Milestones 4 and 5.

3. **Inverting an encryption key**

   Most letter-substitution ciphers require the sender and receiver to use different keys: one to encrypt the message and one to decrypt the message when it reaches its destination. Your task in this exercise is to write a function `invert_key` (which you will also need for the Enigma project) that takes an encryption key and returns the corresponding decryption key. In cryptography, that operation is called *inverting* the encryption key.

   The idea of inverting a key is most easily illustrated by example. Suppose, for example, that the encryption key is `"QWERTYUIOPASDFGHJKLZXCVBNM"`. That key represents the following translation rule:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Q	W	E	R	T	Y	U	I	O	P	A	S	D	F	G	H	J	K	L	Z	X	C	V	B	N	M

   The translation table shows that A maps into Q, B maps into W, C maps into E, and so on. To turn the encryption process around, you have to read the translation table from the bottom to the top, looping to see what letter in the original text would have produced each letter in the encrypted version. For example, if you look for the letter A in the bottom line of the key, you discover that the corresponding letter in the original must have been K. Thus the first letter in your inverted key would be a K. Similarly, the only way to get a B in the encrypted message is to start with an X in the original one, so the second letter in the inverted key would be an X. So the first two entries in the inverted translation table look like:

A	B
↓	↓
K	X

If you continue this process by finding each letter of the alphabet on the bottom of the original translation table and then looking up to see what letter appears above, you will eventually complete the inverted table, as follows:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
K	X	V	M	C	N	O	P	H	Q	.R	S	Z	Y	I	J	A	D	L	E	G	W	B	U	F	T

The inverted key is simply the 26-character string on the bottom row, which in this case is `"KXVMCNOPHQRSZYIJADLEGWBUFT"`.