1 Words as base 26 numbers

By thinking of the letters as numbers, A=0, B=1, C=2, ... , Z=25, we can think of words as numbers written in base 26. For example, the word EXIT corresponds to the number $4 \cdot 26^3 + 23 \cdot 26^2 + 8 \cdot 26 + 19 = 86079$. We are now going to write code that transfers between these representations.

Words to numbers:

We can think of going from the word EXIT to the number 86079 as a process of two steps. First we identify the digits corresponding to the letters; in this case E=4, X=23, I=8 and T=19. Second, we multiply these by powers of 26 and add the results; E is in the 26^3's place, X is in the 26^2's place, I is in the 26's place and T is in the 1's place. Thus we get

$$EXIT \qquad \sim \qquad 4 \quad 23 \quad 8 \quad 19 \qquad \sim \qquad 4 \cdot 26^3 + 23 \cdot 26^2 + 8 \cdot 26 + 19$$

Try your function on the words on 'SABOTAGE' and 'DOUBLEAGENT', you should get 144591196312 and 503702357275045.

It is worth mentioning that you have already written code that does the first of these steps, on the first or second day of this camp.

Numbers to Words:

How does one figure out that 86079 translates back into the word EXIT? One way would be to write the number in base 26 and then read off the digits. That is, if we can figure out that

$$86079 = 4 \cdot 26^3 + 23 \cdot 26^2 + 8 \cdot 26 + 19$$

then we know that the word consists of the 4th, 23rd, 8th and 19th letters of the alphabet, so the word is EXIT.

However, we do not actually have to write the number in base 26 to accomplish this. To see why, let us examine the above example more carefully.

Notice that 19, the last "letter" of the word, is the remainder of 86079 modulo 26. This tells us that the last letter of the word for 86079 is T. If we subtract this remainder and divide by 26, we get

$$3310 = 4 \cdot 26^2 + 23 \cdot 26 + 8$$

The remainder of this new number modulo 26 is 8, which tells us the second to last letter of our word is I. If we subtract the remainder and divide by 26 again we get

$$127 = 4 \cdot 26 + 23$$

The remainder of this new number modulo 26 is 23, which tells us the next letter is X. Subtracting the remainder and dividing by 26 gives

$$4 = 4$$

and we see that the first letter of our word is E.

(1) Write functions for the above two programmes. Call them whatever you want. Test your code on a **LOT** of examples to make sure it works correctly.

2 Public Key Encryption

In order to run the public key encrytion programmes we are going to be studying for the rest of camp, we are going to want functions that do the following things:

- Generate large prime numbers.
- Find generators for large prime mods.
- Find the inverse of a given number in a given mod.
- Compute the prime factorization of a number. Write this so that it counts repeated factors, e.g. 24 should output [2,2,2,3].

Feel free to work first on the one of these that you find most interesting/approachable. In fact, it is probably good to have different people in the same team work on different things.

3 ElGamal

The idea of ElGamal is as follows: you have a message M that you turn into a number (also called M). After doing a Diffie-Hellman key exchange with another party, you have a shared secret number x^{ab}. You encode your message M by computing $M \cdot x^{ab} (\mod p)$ and sending this to the other party. They decrypt the message by multiplying by the inverse of x^{ab}.

One way to implement this is to use your `toNumbers` function to change your message into a list of numbers and then to multiply each number in the list by x^{ab} and reduce mod p. The disadvantage of this method is that all of the A's stay 0 and all of the B's become the secret key x^{ab} which you want to keep a secret.

A more secure method would be to use the functions we have written at the beginning of this lab to change your message into a single number instead of a list of numbers. The difficulty here is that ElGamal will not work if the number your message turns into is larger than the prime mod; the longer your message is the bigger your prime has to be, and large primes are not always so easy to come by. There are two obvious ways to fix this: only send short messages, or start by writing a function that takes your message and breaks it up into two letter chunks and then to write your ElGamal function so that it works on each chunk separately.

Something to keep in mind is that if you want to use ElGamal to send secret messages to someone else then they need to be "doing ElGamal" the same way that you are.

Write a function `ElGamal(message,mod,key)` that implements ElGamal in some fasion. Once you think you've got it working let me know and we can try to send and recieve messages.