README.md

Report of Assignment 1 Part 2

This report contains information on the script `cryptanalysis.py`.

Usage instructions

1. Make sure you have `python3` installed, with the following packages
   - `numpy`
   - `sympy`

   Following commands can achieve that.

   ```
   $ sudo apt install python3 python3-pip
   $ pip3 install numpy sympy
   ```

2. Open `cryptanalysis.py` and scroll to the bottom. Make sure the variable `plaintext` contains the plain text string, and the variable `k` contains the key as a `numpy.array`. There are some prepopulated values but you're free to edit them as needed.
3. Run the script.

   ```
   $ python3 cryptanalysis.py
   ```

The output will be multiple blocks of the following pattern

```
[[ 0 14 19]
 [ 7  7 21]
 [ 7 20  8]]
88.18022012641926
peekiigb od erammi nn f iiurrtiief snsst hhns sojwyg tlkkngg booti isv ipaati it uhl iieerls snsst hhwrrteek repss
trrsssngg no oei vryy mpprttntt ovitg tagt rremss aiva nn mpprttntt oylt oo lnai nnmeeipcnp pliiipca ffeer lll l
iieerls sciityy sf frmmdo ot uhb aaiss fo oefs maainntiint tefsi mmgiiagtonnarra r rsuutb oo ansf fere hboghhsj
wipcc nab re eylteet bo rremss sd denainnp rrviiefs p iituueb ot uhreelw wrllu anovvriigg tiansw wipco gteerissm
iihtt ogt aivb eenp pnddreeu cpnd det tn aarboedd nddizmteef ereddmo oi zmgiiiani tt ayy eeer eekllsss op cnssdeet
uhp oosiiiyltyy ft trnnngg oq denast twookg trboghh hre oyltiiall onnigtonnt ooaly ugt gnnriigg tezm lttgeehrem
vihtt veeb re orrr eekllsssayyt hha uuhboa
```

Each block corresponds to a probable solution of the decryption process. Each block contains

- The decrypted key
- The chi-squared value of the decrypted text, which signifies how close the decrypted text is to the English language. So, the lower the better.
- The decrypted text.

A manual inspection over these blocks can easily identify the block which contains legible English text, and that would be the correct decryption.

Summary of the script and the cryptanalysis strategy involved

Cryptanalysis strategy

If I were to brute force through the space of all possible keys for a given key size, it would be practically impossible to scale it beyond the trivial `2×2` case. So, I've taken a more nuanced approach here that scales beyond that.

Let's assume, the key size is `N×N`.

For each block of letters in the plaintext, one row of the key is used to generate one letter in the corresponding block of ciphertext. Knowing this, I've went through all `26^N` possible rows of the key. For each row, I've multiplied it with the ciphertext block, and so I got one letter out of it. After doing this for the whole ciphertext, the chi-squared value of the resulting plaintext is calculated, and if it's below a particular threshold—I've set this at 150—it's considered for the next step.

In the next step, I have some candidate rows that we got from the previous step, let's consider I have `C` rows. Now all possible keys that can be created using these rows are generated, there will be `Permutation(C, N)` such keys. For each key, calculate the plaintext and then calculate the chi-squared value for that. If the chi-squared value is less than the threshold, output it.

Some specificities of the script

- It uses the `encrypt` method from the `hill.py` file for encryption.
- Spaces in the text are preserved by saving their locations before sending the text into the encryption// decryption pipeline, and then adding those spaces in after it's out of the corresponding pipeline.