TP1 Report

A85272	Jorge Mota
A83840	Maria Silva

This work has the objective to decipher 3 ciphertexts and do the respective cryptanalysis.

The decrypted results are in the files `plaintext1.txt` , `plaintext2.txt` and `plaintext3.txt` .

The ciphertexts are in uppercase characters and plaintexts in lowercase.

This ciphertexts were encrypted with *affine* , *monoalphatic substitution* and *Vigenère* (not respectively). Alhtough we didn't know which method of encryption was aplied in order, we deduced that the first one wasn't the *Vigenère*, because there was a lot of texts and too few patterns for 3 letter words.

With this we could try to first find the *affine* ciphertext since its the easist one to get the key (only 2 numbers obtained from systems of equations with modular aritmethic).

But instead we decided to decypher the *affine* and the *monoalphatic substitution* at the same time , since the *affine* is a type of *monoalphatic substitution*.

Ciphertext #2 & #3

We deciphered this texts with de the `mono_decoder` function we developed in python

This function receives a ciphertext (in uppercase) and optionally a known_letters map and returns the plaintext if its possible to decode

```
mono_decoder(txt,kl={})
```

This implementations follows a simple algorithm:

1. Swap letters in `txt` with the `known_letters` map (this will increase the confidence in the results)

2. While is text not fully decrypted

2.1. Find confidence and matches for every word **2.2.** Select the word with max confidence value

Confidence

Confidence is a value from `0` to `1` that represents the algorithm's confidence to deduce the word and make changes to the `known_letters` map. This value is just a fraction with the number of matches an encrypted word gives.

Match

For an encrypted word 'JGG', swapping uppercase characters for dots (`.`) and passing it to the `match` function developed, it returns a list of possible matched in the `words.txt` file, example:

```

> match('...') ['act', 'add', 'age', 'ago', 'aid', 'aim', 'air', 'all', 'and', 'any', 'are', ... ] ```

This can be done with a partially encrypted word too. For the word `Joo` passing `.oo` will reduce the number of matches in the result, example:
```

```

> match('.oo') ['too ] ```

### Known Letters Map

It's a python dictionary used as a map to store the mapping of the cipher letterto plain letters, example:

```
 known_letters = { 'G':'t', 'U':'a', 'V':'s', 'I':'o', 'T':'r', 'R':'u', 'N':'b', 'O':'l',
'S':'w', 'H':'i', 'J':'p', 'C':'n', 'B':'m', 'M':'c', 'A':'y', 'K':'g', 'F':'v', 'Z':'f',
'L':'d', 'L':'d', 'Y':'q', 'P':'k', }
```

---

# Ciphertext #1

---
```