CodeBreakers Guide

Overview:

CodeBreakers is an application for helping people make and break substitution ciphers. On the title screen are 3 buttons: ‘decode’, ‘encode’, and ‘Credits’. These take the user to the decoding screen, the encoding screen, and the credits screen respectively.

Decode Screen:

Under ‘Enter message:‘ is a space for the user to type in their decrypted message. Hit ‘decrypt’ for the program to decrypt the message! The decrypted message as well as the ciphertext to plaintext alphabet translation will be displayed below.

Note that this was designed for solving monoalphabetic substitution ciphers. The decoder works best for long messages, meaning it’s most effective on single paragraphs or lengthy quotes, 3 sentences minimum. Any letters the decoder cannot deduce will be displayed with ‘_’.

Try copy and pasting these messages into it:

> Sy l nlx sr pyyacao l ylwj eiswi upar lulsxrj isr sxrjsxwjr, ia esmm rwctjsxsza sj wmpramh, lxo txmarr jia aqsoaxwa sr pqaceiamnsxu, ia esmm caytra jp famsaqa sj. Sy, px jia pjiac ilxo, ia sr pyyacao rpnajisxu eiswi lyypcor l calrpx ypc lwjsxu sx lwwpcolxwa jp isr sxrjsxwjr, ia esmm lwwabj sj aqax px jia rmsuijarj aqsoaxwa. Jia pcsusx py nhjir sr agbmlsxao sx jisr elh. -Facjclxo Ctrramm

> f pbb qeb tloia ybfkd piltiv qoxkpclojba fkql x tfiabokbpp; f ebxo qeb xmmolxzefkd qerkabo qexq, lkb axv, tfii abpqolv rp qll. f cbbi qeb prccbofkd lc jfiiflkp. xka vbq, tebk f illh rm xq qeb phv, f pljbelt cbbi qexq bsbovqefkd tfii zexkdb clo qeb ybqqbo, qexq qefp zorbiqv qll pexii bka, qexq mbxzb xka qoxknrfifqv tfii obqrok lkzb jlob. -xkkb coxkh

The decoding is taken care of by the decode module. In short, the module breaks words down into letters patterns and searches a dictionary for all the words that fit those patterns in order to deduce which letters are mapped to which.

> For example, the encrypted word ‘kepp’ has a pattern 0.1.2.2. The words in the dictionary that fit this pattern include ‘pass’, ‘bass’, ‘boss’, and ‘call’. From just this information alone, we’ve deduced that {k:[p,b,c], e:[a,o], p:[s,l]}. By doing this for every word in the message and finding the intersection of their possible letters, we can find letters with only 1 possible mapping. Eliminating solved letters from the maps should create more letters that map to only 1, until

eventually the code is solved. For a more detailed explanation as well as some python code, visit https://inventwithpython.com/hacking/chapter18.html

One thing the decode module does that inventwithpython does not is a final 'clean-up' step. Here, words with missing letters still are filled in with the remaining unmapped letters and then compared to the words in the dictionary. If only one possible letter creates an English word, that letter is mapped. This 'clean-up' procedure is repeated until no further letters can be deduced.

For example, if we have the word 'dkg' mapped to 'SA_', and from earlier it was found that {g:[x,u,q}, the module would look up 'SAX', 'SAU', and 'SAQ' in the dictionary and find that only 'SAX' is an English word. It would then map g to X and precede with the rest of the incomplete words.

The dictionary in the decode module was obtained from inventwithpython. Two-letter words were added to aid in decoding shorter messages.

Encode Screen:

Like the decode screen, there is a space for the user to type in a message, in this case plaintext. At the bottom is a strip where the user can type in the ciphertext alphabet below a plaintext alphabet. The only restriction is that the user does not exceed 26 characters. Users can map multiple different letters to the same letters, or even map letters to symbols. Ciphertext alphabets are not saved, so if a user wishes to use certain alphabets frequently, they should save them within a text file for easy access.

Once everything is inputted, press encrypt to encode the message. The program uses dictionaries within the encode module to reassign the letters of the plaintext message. The encode module actually contains classes for creating well-known ciphertext alphabets including the shift code as well as a random alphabet generator, but sadly there wasn't time to implement these into the program. They're fully functional for importing into other applications, however.

Try encoding something and then placing it into the decoder to see if you receive the original message!

Credit Screen:

Provides the origins of the decoding algorithm as well as the brick background. Thanks for trying out the program!