Week 9 - Monday

COMP 1800

Last time

- What did we talk about last time?
- Work day
- Before that:
  - Function variables
  - Passing functions to other functions

Questions?

Assignment 7

Cryptanalysis

Cryptography

- "Secret writing"
- The art of encoding a message so that its meaning is hidden
- **Cryptanalysis** is breaking those codes
- Now that our Python skills are stronger, we can try to do some cryptanalysis

Encryption and decryption

- **Encryption** is the process of taking a message and encoding it
- **Decryption** is the process of decoding the code back into a message
- A **plaintext** is a message before encryption
- A **ciphertext** is the message in encrypted form
- A **key** is an extra piece of information used in the encryption process

Transposition cipher

- In a transposition cipher, the letters are reordered but their values are not changed
- Any transposition cipher is a permutation function of some kind

Example: Rail Fence Cipher

- In the rail fence cipher, a message is written vertically along a fixed number of "rails," wrapping back to the top when the bottom is reached
- To finish the encryption, the message is stored horizontally
- This is also known as a **columnar transposition**
- Encryption of "WE ARE DISCOVERED, FLEE AT ONCE" with three rails:

W	R	I	O	R	F	E	O	E
E	E	S	V	E	L	A	N	X
A	D	C	E	D	E	T	C	J

- **Ciphertext:** WRIORFEOEEESVELANXADCEDETCJ

Rail fence encryption

- Several chapters ago, our attempt at rail fence encryption was only an even-odd shuffle
- Now, let's write a function to do a full rail fence encryption with an arbitrary number of rails
- We need proper encryption and decryption functions if we want to do cryptanalysis

Rail fence algorithm

```
def railEncrypt(plaintext, number):
```

- Create a list holding **`number`** empty strings
- Iterate over all the characters in **`plaintext`**
  - Use a counter to decide which string in the list to concatenate the character onto
    - Hint: The modulus operator lets us wrap around easily
- Concatenate all the strings together
- Note: There are problems if the length of the plaintext isn't evenly divisible by the number of rows
  - Typically, random values are added to pad out the plaintext

Python to make rail fence encryption easier

- Although it's not hard to concatenate all the rails together, there is a Python tool designed for making a string out of everything in a list
  - This tool can also be useful for the ghostwriter project
- String objects have a **`join()`** method that will join a list together into a string, using the string as a separator

```
words = ['my', 'dog', 'has', 'fleas']
result = ''.join(words)    # 'mydoghasfleas'
result = ' '.join(words)   # 'my dog has fleas'
result = '|'.join(words)   # 'my|dog|has|fleas'
result = 'pig'.join(words) # 'mypigdogpighaspigfleas'
```

Rail fence decryption

- A little bit of math is useful when doing the rail fence decryption
- Consider where the characters end up from the original plaintext based on the rails

	Columns							
	0	1	2	3	4	5	6	7
Row 0	0	3	6	9	12	15	18	21
Row 1	1	4	7	10	13	16	19	22
Row 2	2	5	8	11	14	17	20	23

- The character in location $(row, column)$ can be found at index $(row \cdot length + column)$ where $length$ is the length of a rail

Rail fence decryption

```
def railDecrypt(ciphertext, number):
```

- Determine how long the rows are
- Loop over all the columns
  - Loop over all the rows
    - Use the formula $(row \cdot length + column)$ to get the next character in the output
    - Concatenate this character to your output string
- Return the output, split into a list of strings

Brute force cryptanalysis

- **Brute force** means trying all possibilities
- For some kinds of encryption, that would mean trying trillions of possibilities
- For a rail fence cipher, the possible numbers of rails go from 2 up to the length of the message
- Thus, we can make a simple brute force function that runs our decryption algorithm with all possible rail sizes

```python
def railBrute(ciphertext):
    for i in range(2, len(ciphertext) + 1):
        print(railDecrypt(ciphertext, i))
```

Automated brute force

- Although the previous function gets the right answer, we have to look at all the encryptions to see which one makes sense
- However, if we load a file containing English words into a Python dictionary, we could see how many real words show up in each decryption
- Then, we could store the one with the most real English words, assuming that is the best decryption

Loading words into a dictionary

```
def loadWords(filename):
```

- Create an empty dictionary
- Open the file called **`filename`**
- Loop over all the lines in the file
  - Put each one into the dictionary, with a value of **`True`**
  - Be sure to clean off the last character of the word (or use the **`strip()`** method to remove whitespace)
- Return the dictionary

- Note: This function only works with a file that contains a single word on each line
- The value of **`True`** is unimportant, we just want to know whether each word is in the dictionary, and looking up values in a dictionary is faster than a list

Automated brute force

- Now that we can load the dictionary, we can make an automated brute force function:

```
def railAutomated(ciphertext):
```

- Load the dictionary
- Create a variable for the highest number of words found in a decrypted phrase
- Create a variable for the best decrypted phrase
- Loop over possible rail lengths:
    - Decrypt with the given length
    - Loop over the words in the decrypted list and count how many are in the dictionary
    - If there are more in the dictionary than the highest
        - Update the highest count and the best decrypted phrase
- Return the best decrypted phrase

A few observations

- This automated approach only works because the encrypted phrase has spaces in it
- It's not difficult to improve this approach to work even if there are no spaces in the message
  - But the code is much uglier
- The small number of possible rails (which is what makes the key) makes it easy to brute force a rail fence cipher

Cryptanalysis of Substitution Ciphers

Simple monoalphabetic substitution cipher

- We can map to a random permutation of letters
- For example:

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

- E("MATH IS GREAT") = "UIYP TQ ABZIY"
- 26! possible permutations
- Hard to check every one

Upcoming

Next time…

- More on cryptanalysis
- Doing frequency analysis in Python

Reminders

- Read 8.4 for Wednesday
- **Work on Assignment 7**