Let's get started! Create a new file in a project in Eclipse or LiClipse and name it encrypt.py. Be sure to include the standard code header, and add some comments!:

# Author: # Pair Partner: ######################################################################## # Lab 7: Encryption ########################################################################

You'll be writing at least **four functions** this week. You may write other "helper" functions, and you can also write code to use/test those functions, as you did last week, but all of the encrypting and decrypting should happen when you call these functions.

In all of these functions, the argument s will be a string containing lowercase letters, punctuation, and spaces.

* **Function #1: random\_encipher( s,n )**  
    
  Your arguments for this function will be s as described above, and some seed value n. Your function should return a new string in which the letters have been encoded using the substitution cipher I'll describe below. Use the seed value as described in the Seeding the Random Number Generator section above, to get a predictable sequence of results.  
    
  This function will create a random substitution cipher. First, you'll want to create a random ordering of the letters of the alphabet. I recommend using random.shuffle(), but be careful! This only works on *lists*.  
    
  Now, you'll be determining the substitution by treating your current letter as a number - "a" == 0, for example, which means that in order to encode "a" you'll be looking at the zero-index element of your shuffled alphabet.  
    
  There's an easy way to convert letters to numbers, by the way - use the built-in functionord(letter). If you call ord("a"), for example, you'll get the int value 97; ord("b") == 98, and so on. How could you use this to make "a" == 0, "b" == 1, and so on??

def **random\_encipher(s,n):** # seed the random number generator # shuffle the alphabet # build the encoded string by using the decoded letter # as an index into your shuffled alphabet # return the encoded string

* **Function #2: random\_decipher( s,n )**  
    
  What good is a cipher if you can't decode it? This function should reverse the random\_encipherfunction using the seed value n and return the decoded string. Use what you know about the shuffled alphabet you generated in random\_encipher to help you translate the string back.

def **random\_decipher(s,n):** # seed the random number generator # shuffle the alphabet # build the decoded string by reversing your process # from random\_encipher # return the decoded string

* **Function #3: rotate\_encipher( s,n )**  
    
  Your arguments for this function will be s as described above, and a non-negative integer nbetween 0 and 25. Your function should return a new string in which the letters in s have been "rotated" forward by n characters in the alphabet, wrapping around (that is, "z"becomes "a" when rotated by 1) if necessary.  
    
  All non-alphabetic characters should be left unchanged!

def **rotate\_encipher(s,n):** # build the encoded string by rotating each # letter forward by n spaces - make sure you # wrap around when you hit z! # return the encoded string

**Hint:** try writing another function, rot(c,n) that rotates a single character c forward by nspots in the alphabet. Remember you'll need to "wrap around" and *leave non-alphabetic characters unchanged*. If you write this function, here are some test cases:

rot('a',2) --> 'c' rot('y',2) --> 'a' rot(' ',4) --> ' '

How might this be useful in writing your rotate\_encipher function?

* **Function #4: rotate\_decipher( s )**  
    
  Again we're decoding a coded message, but this time we don't have a key - we'll have to guess. We do have the advantage here, though, of knowing *how* the message was encoded - by rotation - so we can narrow down the 403291461126605635584000000 possible substitution ciphers to a mere 26 rotations!  
    
  Your function should return, to the best of its ability, the un-rotated English version of the argument string s.

def **rotate\_decipher(s):** # find all of the possible encodings # score each encoding on its "englishness" # find the highest-scoring encoding # return the corresponding string

**Hint #1:** start out by finding all of the possible encodings. Do you have some function that could help you do that??

all\_encodings = [ \_\_\_\_ for n in range(26) ]

**Hint #2:** it's up to you how you'd like to score how "English" each of these encodings is, but you'll want to find the *most* English-looking of the possibilities and return that as your decoded message. Here's a possible function you can copy and paste into your own code directly, if you like - or feel free to make up your own:

def letProb( c ): """ if c is the space character or an alphabetic character, we return its monogram probability (for english), otherwise we return 1.0 - we ignore capitalization. Adapted from http://www.cs.chalmers.se/Cs/Grundutb/Kurser/krypto/en\_stat.html """ if c == ' ': return 0.1904 if c == 'e': return 0.1017 if c == 't': return 0.0737 if c == 'a': return 0.0661 if c == 'o': return 0.0610 if c == 'i': return 0.0562 if c == 'n': return 0.0557 if c == 'h': return 0.0542 if c == 's': return 0.0508 if c == 'r': return 0.0458 if c == 'd': return 0.0369 if c == 'l': return 0.0325 if c == 'u': return 0.0228 if c == 'm': return 0.0205 if c == 'c': return 0.0192 if c == 'w': return 0.0190 if c == 'f': return 0.0175 if c == 'y': return 0.0165 if c == 'g': return 0.0161 if c == 'p': return 0.0131 if c == 'b': return 0.0115 if c == 'v': return 0.0088 if c == 'k': return 0.0066 if c == 'x': return 0.0014 if c == 'j': return 0.0008 if c == 'q': return 0.0008 if c == 'z': return 0.0005 return 1.0

**SAMPLE OUTPUT: SHUFFLE**

Testing random\_encipher:

random\_encipher("caesar cipher? i prefer caesar salad.",25) --> rdmsdw rkgpmw? k gwmcmw rdmsdw sdzdv. random\_encipher("to infinity, and beyond!",-4) --> se hyxhyhsc, ayl fqceyl!

Testing random\_decipher:

random\_decipher("rdmsdw rkgpmw? k gwmcmw rdmsdw sdzdv.",25) --> caesar cipher? i prefer caesar salad. random\_decipher("se hyxhyhsc, ayl fqceyl!",-4) --> to infinity, and beyond!

**SAMPLE OUTPUT: SHIFT**

Testing rotate\_encipher:

rotate\_encipher("xyza",1) --> yzab rotate\_encipher("z a",1) --> a b rotate\_encipher("\*ab?",1) --> \*bc? rotate\_encipher("this is a string!",1) --> uijt jt b tusjoh!

Testing rotate\_decipher (note that your results may vary - and may not be perfect! - but for long sentences your function should be pretty reliable):

rotate\_decipher("bzdrzq bhogdq? h oqdedq bzdrzq rzkzc.") --> caesar cipher? i prefer caesar salad. rotate\_decipher("xs mrjmrmxc, erh ficsrh!") --> to infinity, and beyond! rotate\_decipher("onyx balks") --> ihsr vufem (???)

What's up with that last one? Well, that's what *my* implementation of rotate\_decipher thought the translation should have been. It's not perfect, it's just looking for the most typically-English-looking translation. Again, don't worry if your version isn't perfect, as long as it's working on longer phrases!