In [ ]:

```
# Name: <your name here>

import string
from permutations import get_permutations

### HELPER CODE ###
def load_words(file_name):
    '''
    file_name (string): the name of the file containing
    the list of words to load

    Returns: a list of valid words. Words are strings of lowercase letters.

    Depending on the size of the word list, this function may
    take a while to finish.
    '''

    print("Loading word list from file...")
    # inFile: file
    inFile = open(file_name, 'r')
    # wordlist: list of strings
    wordlist = []
    for line in inFile:
        wordlist.extend([word.lower() for word in line.split(' ')])
    print("  ", len(wordlist), "words loaded.")
    return wordlist

def is_word(word_list, word):
    '''
    Determines if word is a valid word, ignoring
    capitalization and punctuation

    word_list (list): list of words in the dictionary.
    word (string): a possible word.

    Returns: True if word is in word_list, False otherwise

    Example:
    >>> is_word(word_list, 'bat') returns
    True
    >>> is_word(word_list, 'asdf') returns
    False
    '''
    word = word.lower()
    word = word.strip(" !@#$%^&*()-_+={}[]|\:;'<>?,./\"")
    return word in word_list


### END HELPER CODE ###
```

In [ ]:

```
WORDLIST_FILENAME = 'words.txt'

# you may find these constants helpful
VOWELS_LOWER = 'aeiou'
VOWELS_UPPER = 'AEIOU'
CONSONANTS_LOWER = 'bcdfghjklmnpqrstvwxyz'
CONSONANTS_UPPER = 'BCDFGHJKLMNPQRSTVWXYZ'

class SubMessage(object):
    def __init__(self, text):
        '''
        Initializes a SubMessage object

        text (string): the message's text

        A SubMessage object has two attributes:
            self.message_text (string, determined by input text)
            self.valid_words (list, determined using helper function load_words)
        '''
        pass #delete this line and replace with your code here

    def get_message_text(self):
        '''
        Used to safely access self.message_text outside of the class

        Returns: self.message_text
        '''
        pass #delete this line and replace with your code here

    def get_valid_words(self):
        '''
        Used to safely access a copy of self.valid_words outside of the class.
        This helps you avoid accidentally mutating class attributes.

        Returns: a COPY of self.valid_words
        '''
        pass #delete this line and replace with your code here

    def build_transpose_dict(self, vowels_permutation):
        '''
        vowels_permutation (string): a string containing a permutation of vowels (a,

        Creates a dictionary that can be used to apply a cipher to a letter.
        The dictionary maps every uppercase and lowercase letter to an
        uppercase and lowercase letter, respectively. Vowels are shuffled
        according to vowels_permutation. The first letter in vowels_permutation
        corresponds to a, the second to e, and so on in the order a, e, i, o, u.
        The consonants remain the same. The dictionary should have 52
        keys of all the uppercase letters and all the lowercase letters.

        Example: When input "eaiuo":
        Mapping is a->e, e->a, i->i, o->u, u->o
        and "Hello World!" maps to "Hallu Wurld!"

        Returns: a dictionary mapping a letter (string) to
        another letter (string).
        '''

        pass #delete this line and replace with your code here

    def apply_transpose(self, transpose_dict):
```

```
        '''
        transpose_dict (dict): a transpose dictionary

        Returns: an encrypted version of the message text, based
        on the dictionary
        '''

        pass #delete this line and replace with your code here
```

In [ ]:
```
class EncryptedSubMessage(SubMessage):
    def __init__(self, text):
        '''
        Initializes an EncryptedSubMessage object

        text (string): the encrypted message text

        An EncryptedSubMessage object inherits from SubMessage and has two attribute
            self.message_text (string, determined by input text)
            self.valid_words (list, determined using helper function load_words)
        '''
        pass #delete this line and replace with your code here

    def decrypt_message(self):
        '''
        Attempt to decrypt the encrypted message

        Idea is to go through each permutation of the vowels and test it
        on the encrypted message. For each permutation, check how many
        words in the decrypted text are valid English words, and return
        the decrypted message with the most English words.

        If no good permutations are found (i.e. no permutations result in
        at least 1 valid word), return the original string. If there are
        multiple permutations that yield the maximum number of words, return any
        one of them.

        Returns: the best decrypted message

        Hint: use the function from permutations
        '''
        pass #delete this line and replace with your code here
```

In [ ]:
```
if __name__ == '__main__':

    # Example test case
    message = SubMessage("Hello World!")
    permutation = "eaiuo"
    enc_dict = message.build_transpose_dict(permutation)
    print("Original message:", message.get_message_text(), "Permutation:", permutati
    print("Expected encryption:", "Hallu Wurld!")
    print("Actual encryption:", message.apply_transpose(enc_dict))
    enc_message = EncryptedSubMessage(message.apply_transpose(enc_dict))
    print("Decrypted message:", enc_message.decrypt_message())

    #TODO: WRITE YOUR TEST CASES HERE
```