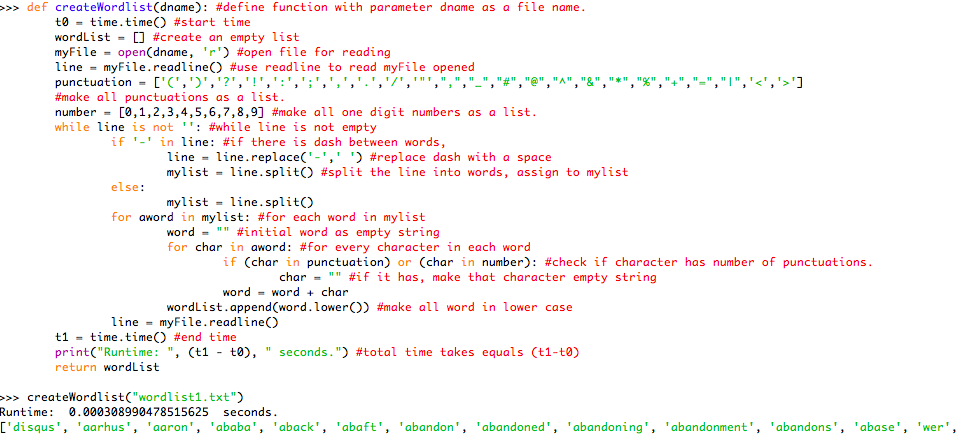
**PPiC 8.3** First, I will write three functions createWordList, createWordSet, and createWordDictionary. Set starting time as t0=time.time() at the beginning of the function. The end time is t1=time.time() at the end of the function. The time it takes simply equals (t1 – t0). This is the same for all three functions.





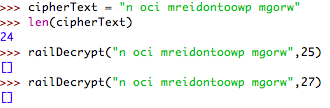


**PPiC 8.9** Now that you have seen the details of the railDecrypt function you can make it smarter in two ways:

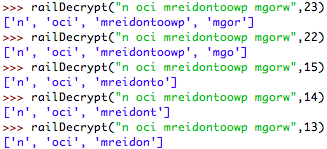
(a) You do not need to check cases where the number of rails is greater than the message length divided by two. Can you explain why?

(b) You only need to check cases where the number of rails evenly divides the total message length. Can you explain why?

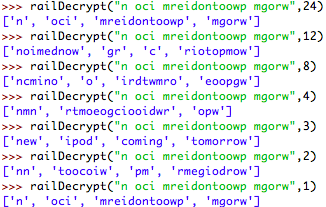
1. Because if the number of rails is greater than the message length divided by two, the railLen = len(cipherText)//numRails will be either 1 or 0. Thus col will be either 0, or 1. So if numRails > len(cipherText), railLen = 0, the decrypted message will be an empty list [].



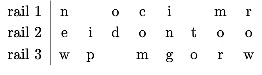
If len(cipherText) > numRails > len(cipherText)//2, railLen = 1, the decrypted message will just simply split the first numRails of ciperText.

****

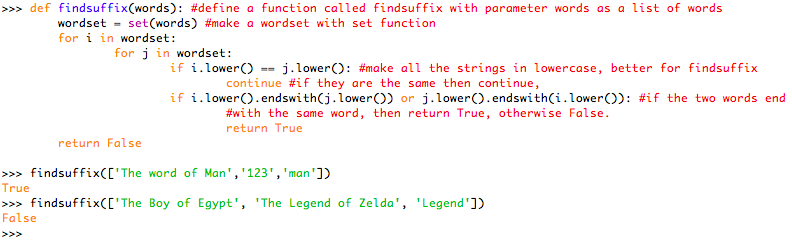
1. Only in cases where len(cipherText) can be evenly divided by numRails, can the cipherText be completely decrypted.



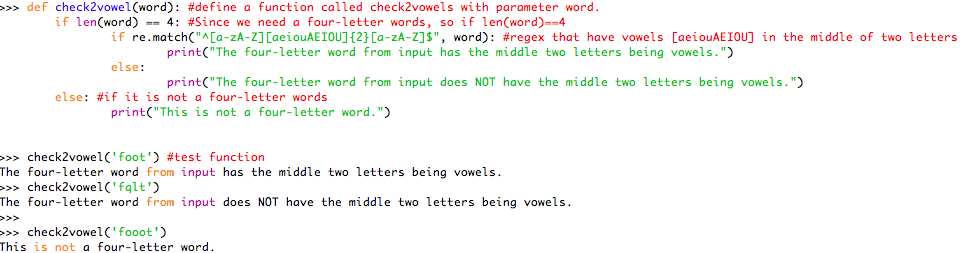
Since the column and row can be specified position = column + row \* rowLendgth, all the space and letters in the cipherText can have a position (position of nextLetter = (col + rail \* railLen)) in the row and column major storage, as the example in picture below.



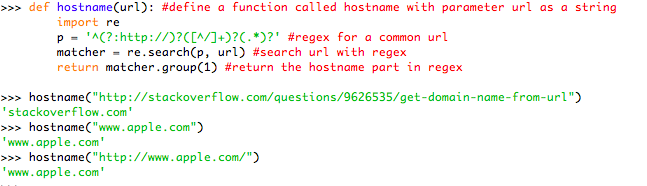
**PPiC 8.16** Write a function that ﬁnds the most popular suffixes for words. You may want to try this function for two- and three-letter suffixes.



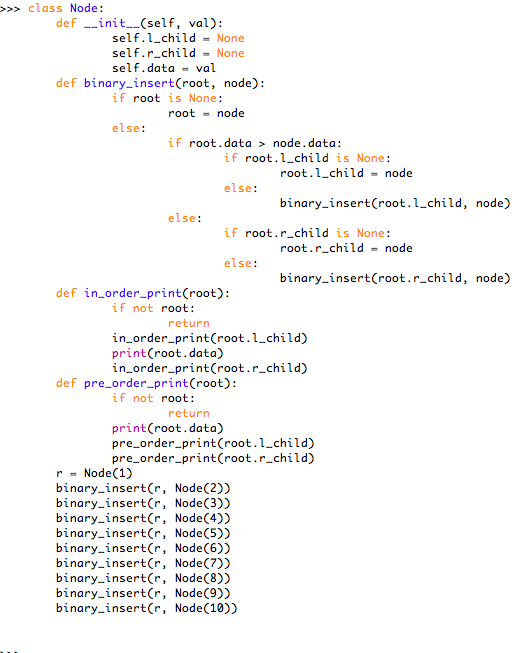
**PPiC 8.22** Write a regular expression to match all the four-letter words where the middle two letters are vowels.

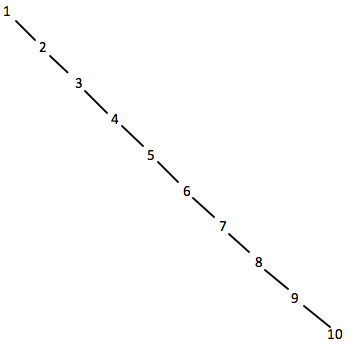


**PPiC 8.23** Write a function that can extract the host name from a URL. The host name is the part of the URL that comes after http:// but before the next /.

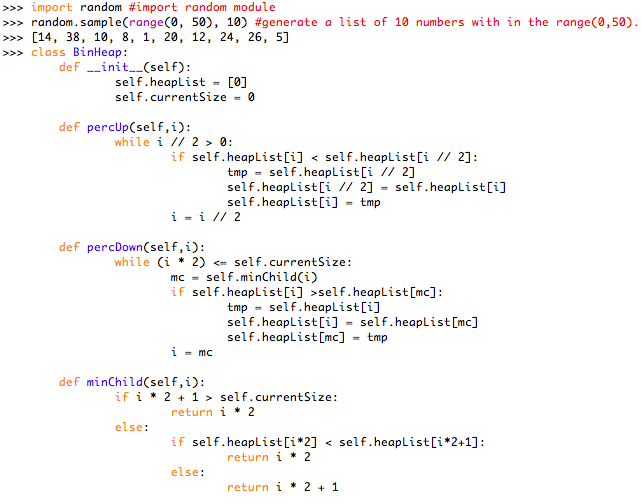


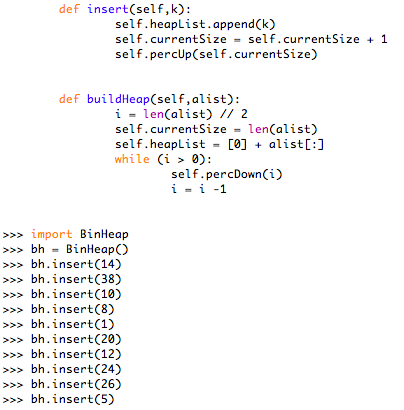
**PS 6.11.3** Consider the following list of integers: [1,2,3,4,5,6,7,8,9,10]. Show the binary search tree resulting from inserting the integers in the list.

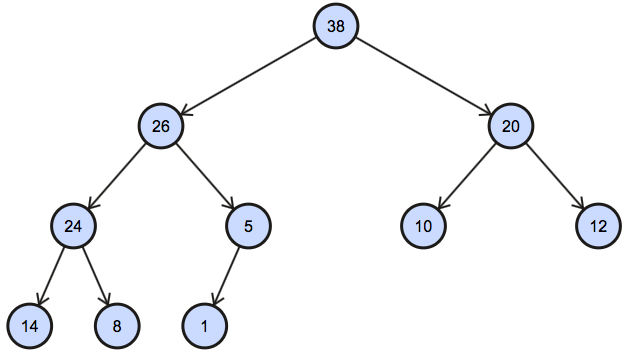




**PS 6.11.5** Generate a random list of integers. Show the binary heap tree resulting from inserting the integers on the list one at a time.







**PS 6.11.8** Generate a random list of integers. Draw the binary search tree resulting from inserting the integers on the list.

