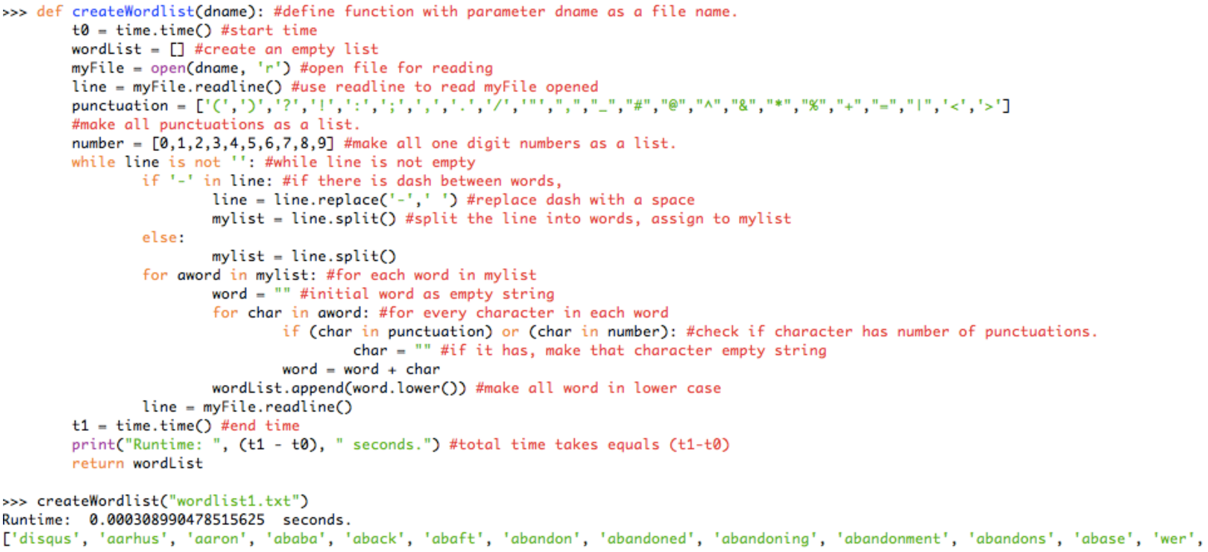
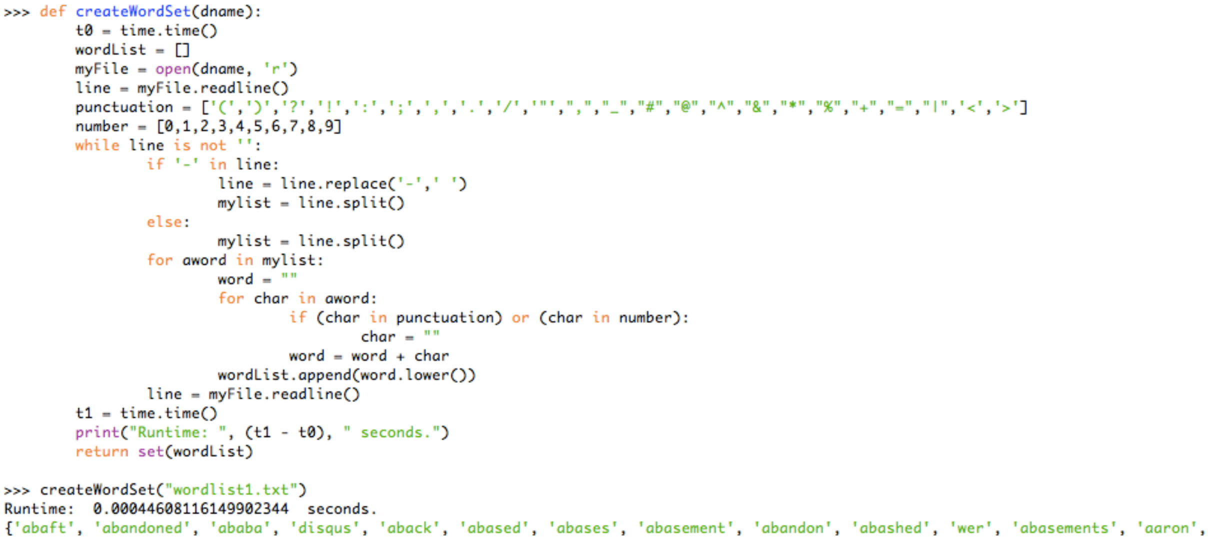
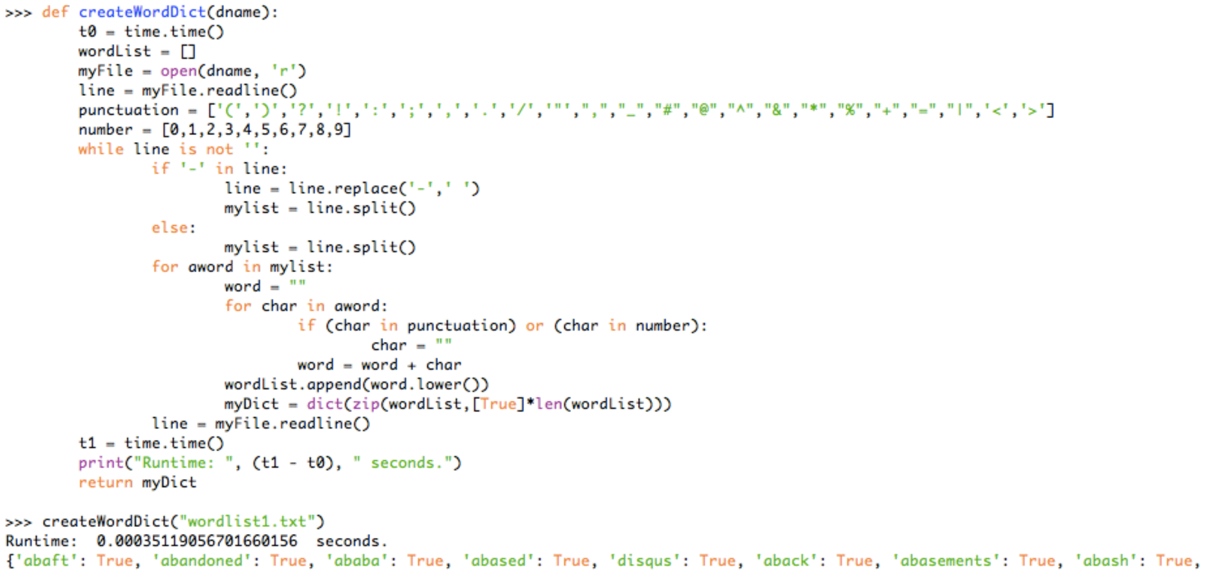
PPiC 8.3 First, I will write three functions createWordList, createWordSet, and createWordDictionary. Set starting time as t0=time.time() at the beginning of the function. The end time is t1=time.time() at the end of the function. The time it takes simply equals (t1 – t0). This is the same for all three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



```
>>> def createWordlist(dname): #define function with parameter dname as a file name.
        t0 = time.time() #start time
        wordList = [] #create an empty list
        myFile = open(dname, 'r') #open file for reading
        line = myFile.readline() #use readline to read myFile opened
        punctuation = ['(',')','?','!',':',';',',','.','/','"',",","_","#","@","^","&","*","%","+","=","|",'<','>']
        #make all punctuations as a list.
        number = [0,1,2,3,4,5,6,7,8,9] #make all one digit numbers as a list.
        while line is not '': #while line is not empty
                if '-' in line: #if there is dash between words,
                        line = line.replace('-',' ') #replace dash with a space
                        mylist = line.split() #split the line into words, assign to mylist
                else:
                        mylist = line.split()
                for aword in mylist: #for each word in mylist
                        word = "" #initial word as empty string
                        for char in aword: #for every character in each word
                                if (char in punctuation) or (char in number): #check if character has number of punctuations.
                                        char = "" #if it has, make that character empty string
                                word = word + char
                        wordList.append(word.lower()) #make all word in lower case
                line = myFile.readline()
        t1 = time.time() #end time
        print("Runtime: ", (t1 - t0), " seconds.") #total time takes equals (t1-t0)
        return wordList

>>> createWordlist("wordlist1.txt")
Runtime:  0.0003089904785l5625  seconds.
['disqus', 'aarhus', 'aaron', 'ababa', 'aback', 'abaft', 'abandon', 'abandoned', 'abandoning', 'abandonment', 'abandons', 'abase', 'wer',
```



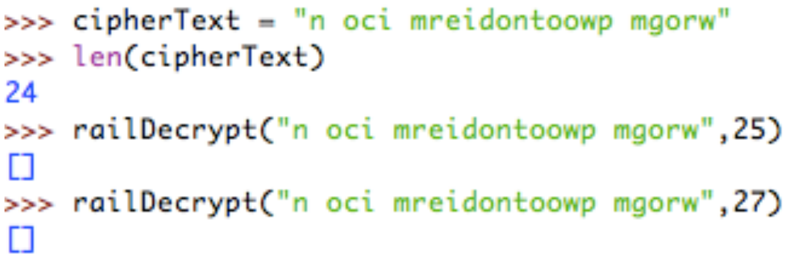
```
>>> def createWordSet(dname):
        t0 = time.time()
        wordList = []
        myFile = open(dname, 'r')
        line = myFile.readline()
        punctuation = ['(',')','?','!',':',';',',','.','/','"',",","_","#","@","^","&","*","%","+","=","|",'<','>']
        number = [0,1,2,3,4,5,6,7,8,9]
        while line is not '':
                if '-' in line:
                        line = line.replace('-',' ')
                        mylist = line.split()
                else:
                        mylist = line.split()
                for aword in mylist:
                        word = ""
                        for char in aword:
                                if (char in punctuation) or (char in number):
                                        char = ""
                                word = word + char
                        wordList.append(word.lower())
                line = myFile.readline()
        t1 = time.time()
        print("Runtime: ", (t1 - t0), " seconds.")
        return set(wordList)

>>> createWordSet("wordlist1.txt")
Runtime:  0.00044608116149902344  seconds.
{'abaft', 'abandoned', 'ababa', 'disqus', 'aback', 'abased', 'abases', 'abasement', 'abandon', 'abashed', 'wer', 'abasements', 'aaron',
```



```
>>> def createWordDict(dname):
	t0 = time.time()
	wordList = []
	myFile = open(dname, 'r')
	line = myFile.readline()
	punctuation = ['(',')','?','!',':',';',',','.','/','"',",","_","#","@","^","&","*","%","+","=","|",'<','>']
	number = [0,1,2,3,4,5,6,7,8,9]
	while line is not '':
		if '-' in line:
			line = line.replace('-',' ')
			mylist = line.split()
		else:
			mylist = line.split()
		for aword in mylist:
			word = ""
			for char in aword:
				if (char in punctuation) or (char in number):
					char = ""
				word = word + char
			wordList.append(word.lower())
			myDict = dict(zip(wordList,[True]*len(wordList)))
		line = myFile.readline()
	t1 = time.time()
	print("Runtime: ", (t1 - t0), " seconds.")
	return myDict

>>> createWordDict("wordlist1.txt")
Runtime:  0.00035119056701660156  seconds.
{'abaft': True, 'abandoned': True, 'ababa': True, 'abased': True, 'disqus': True, 'aback': True, 'abasements': True, 'abash': True,
```

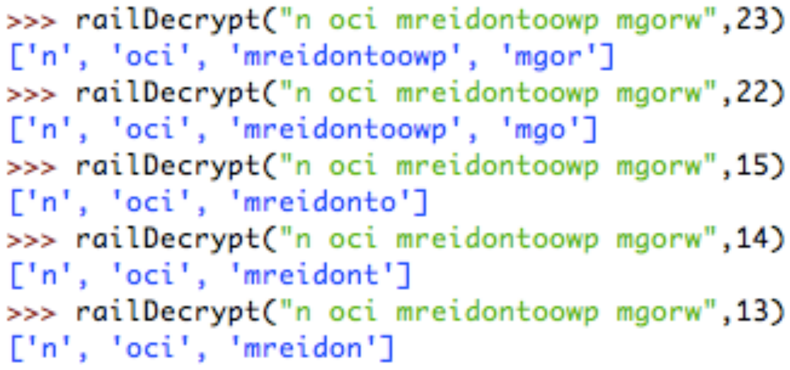
PPiC 8.9 Now that you have seen the details of the `railDecrypt` function you can make it smarter in two ways:
(a) You do not need to check cases where the number of rails is greater than the message length divided by two. Can you explain why?
(b) You only need to check cases where the number of rails evenly divides the total message length. Can you explain why?

(a) 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 [].



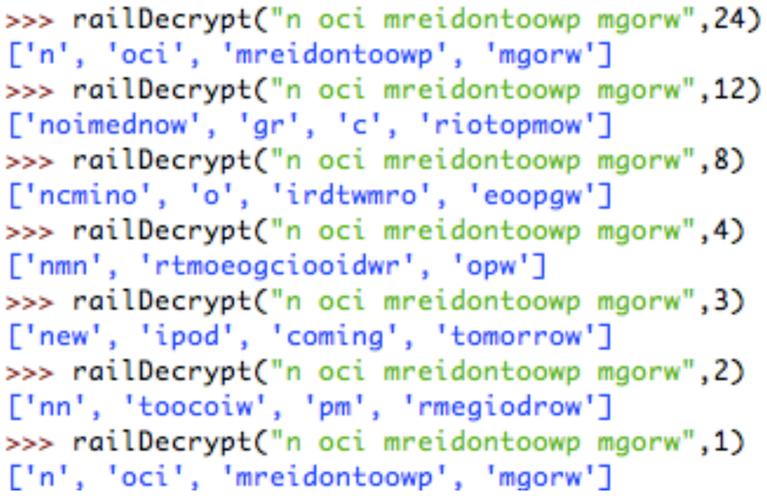
```
>>> cipherText = "n oci mreidontoowp mgorw"
>>> len(cipherText)
24
>>> railDecrypt("n oci mreidontoowp mgorw",25)
[]
>>> railDecrypt("n oci mreidontoowp mgorw",27)
[]
```

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



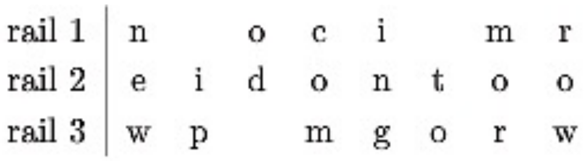
```
>>> railDecrypt("n oci mreidontoowp mgorw",23)
['n', 'oci', 'mreidontoowp', 'mgor']
>>> railDecrypt("n oci mreidontoowp mgorw",22)
['n', 'oci', 'mreidontoowp', 'mgo']
>>> railDecrypt("n oci mreidontoowp mgorw",15)
['n', 'oci', 'mreidonto']
>>> railDecrypt("n oci mreidontoowp mgorw",14)
['n', 'oci', 'mreidont']
>>> railDecrypt("n oci mreidontoowp mgorw",13)
['n', 'oci', 'mreidon']
```

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



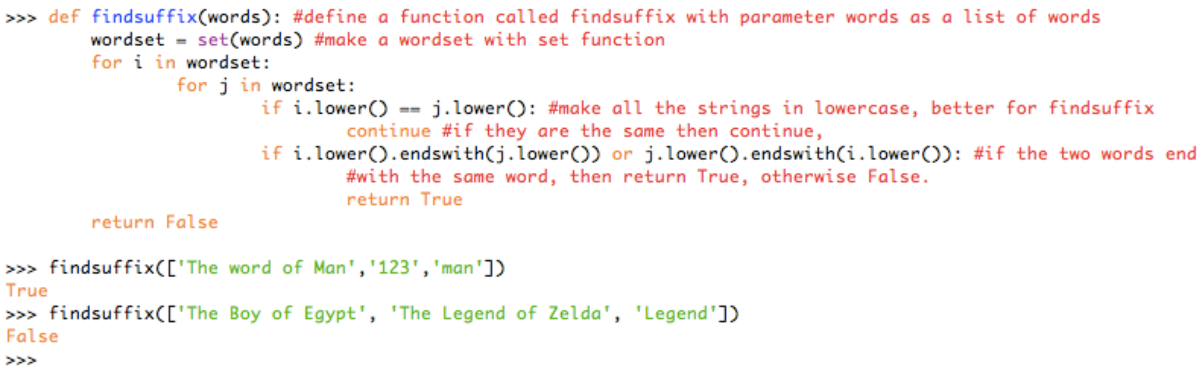
```
>>> railDecrypt("n oci mreidontoowp mgorw",24)
['n', 'oci', 'mreidontoowp', 'mgorw']
>>> railDecrypt("n oci mreidontoowp mgorw",12)
['noimednow', 'gr', 'c', 'riotopmow']
>>> railDecrypt("n oci mreidontoowp mgorw",8)
['ncmino', 'o', 'irdtwmro', 'eoopgw']
>>> railDecrypt("n oci mreidontoowp mgorw",4)
['nmn', 'rtmoeogciooidwr', 'opw']
>>> railDecrypt("n oci mreidontoowp mgorw",3)
['new', 'ipod', 'coming', 'tomorrow']
>>> railDecrypt("n oci mreidontoowp mgorw",2)
['nn', 'toocoiw', 'pm', 'rmegiodrow']
>>> railDecrypt("n oci mreidontoowp mgorw",1)
['n', 'oci', 'mreidontoowp', 'mgorw']
```

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.



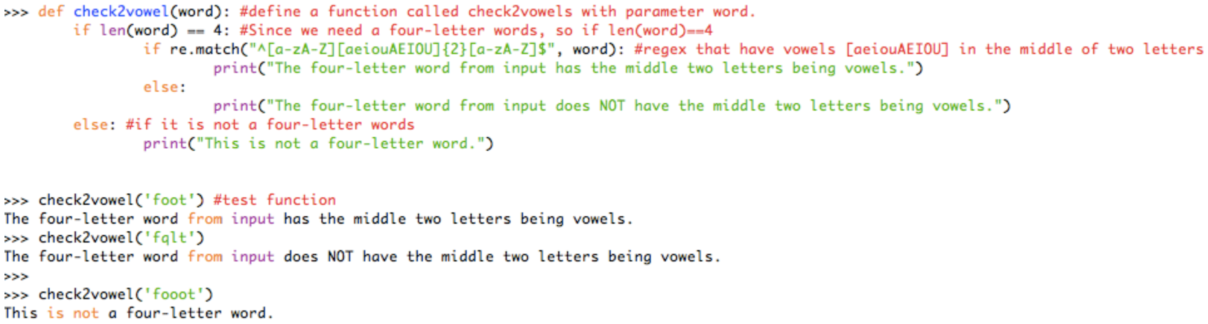
rail 1	n		o	c	i		m	r
rail 2	e	i	d	o	n	t	o	o
rail 3	w	p		m	g	o	r	w

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.



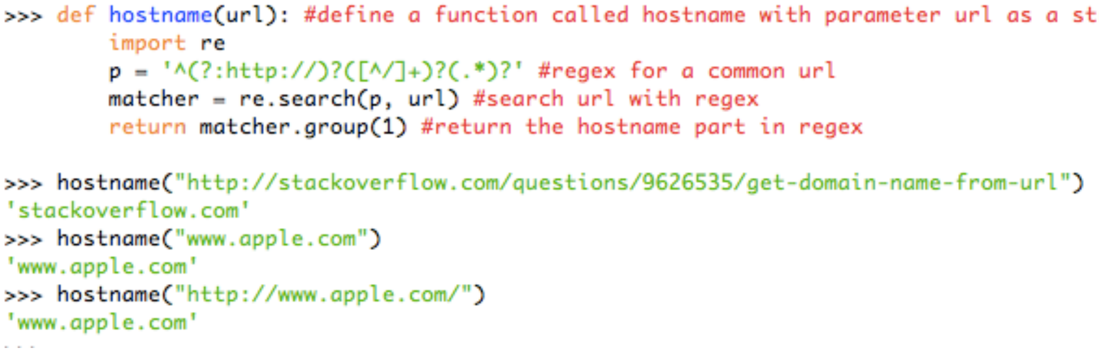
```
>>> def findsuffix(words): #define a function called findsuffix with parameter words as a list of words
	wordset = set(words) #make a wordset with set function
	for i in wordset:
		for j in wordset:
			if i.lower() == j.lower(): #make all the strings in lowercase, better for findsuffix
				continue #if they are the same then continue,
			if i.lower().endswith(j.lower()) or j.lower().endswith(i.lower()): #if the two words end
				#with the same word, then return True, otherwise False.
				return True
	return False

>>> findsuffix(['The word of Man','123','man'])
True
>>> findsuffix(['The Boy of Egypt', 'The Legend of Zelda', 'Legend'])
False
>>>
```

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



```
>>> def check2vowel(word): #define a function called check2vowels with parameter word.
	if len(word) == 4: #Since we need a four-letter words, so if len(word)==4
		if re.match("^[a-zA-Z][aeiouAEIOU]{2}[a-zA-Z]$", word): #regex that have vowels [aeiouAEIOU] in the middle of two letters
			print("The four-letter word from input has the middle two letters being vowels.")
		else:
			print("The four-letter word from input does NOT have the middle two letters being vowels.")
	else: #if it is not a four-letter words
		print("This is not a four-letter word.")


>>> check2vowel('foot') #test function
The four-letter word from input has the middle two letters being vowels.
>>> check2vowel('fqlt')
The four-letter word from input does NOT have the middle two letters being vowels.
>>>
>>> check2vowel('fooot')
This is not a four-letter word.
```

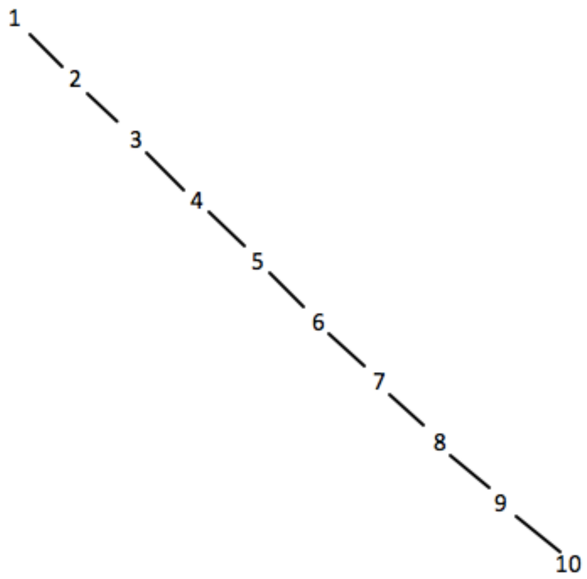
PPiC 8.23 Write a function that can extract the host name from a URL. The host name is the part of the URL that comes after http:// but before the next /.



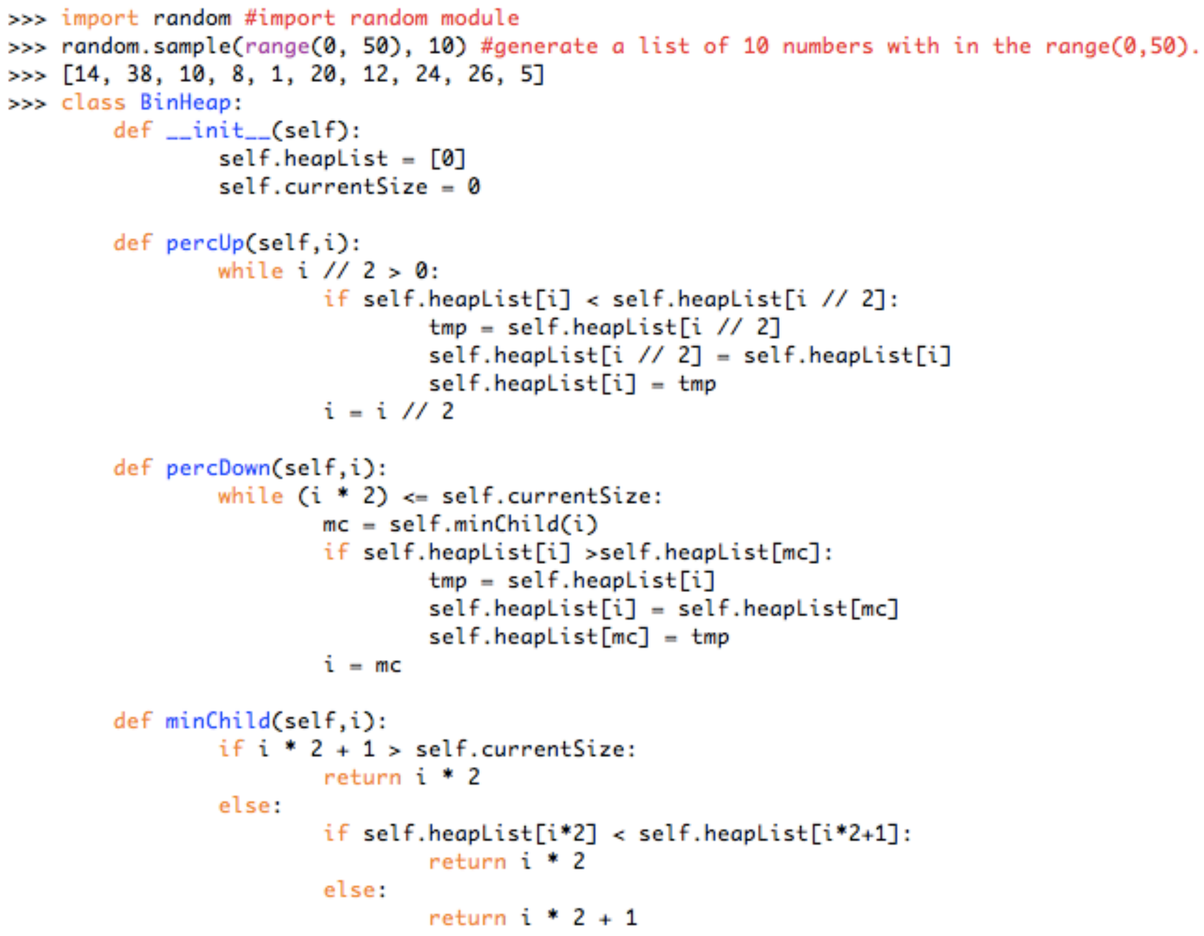
```
>>> def hostname(url): #define a function called hostname with parameter url as a string
	import re
	p = '^(?:http://)?([^/]+)?(.*)?' #regex for a common url
	matcher = re.search(p, url) #search url with regex
	return matcher.group(1) #return the hostname part in regex

>>> hostname("http://stackoverflow.com/questions/9626535/get-domain-name-from-url")
'stackoverflow.com'
>>> hostname("www.apple.com")
'www.apple.com'
>>> hostname("http://www.apple.com/")
'www.apple.com'
```

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.

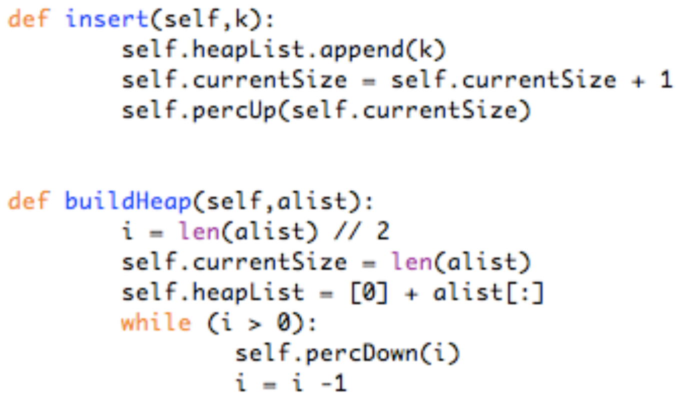
~~~
>>> class Node:
        def __init__(self, val):
                self.l_child = None
                self.r_child = None
                self.data = val
        def binary_insert(root, node):
                if root is None:
                        root = node
                else:
                        if root.data > node.data:
                                if root.l_child is None:
                                        root.l_child = node
                                else:
                                        binary_insert(root.l_child, node)
                        else:
                                if root.r_child is None:
                                        root.r_child = node
                                else:
                                        binary_insert(root.r_child, node)
        def in_order_print(root):
                if not root:
                        return
                in_order_print(root.l_child)
                print(root.data)
                in_order_print(root.r_child)
        def pre_order_print(root):
                if not root:
                        return
                print(root.data)
                pre_order_print(root.l_child)
                pre_order_print(root.r_child)
        r = Node(1)
        binary_insert(r, Node(2))
        binary_insert(r, Node(3))
        binary_insert(r, Node(4))
        binary_insert(r, Node(5))
        binary_insert(r, Node(6))
        binary_insert(r, Node(7))
        binary_insert(r, Node(8))
        binary_insert(r, Node(9))
        binary_insert(r, Node(10))


...
~~~



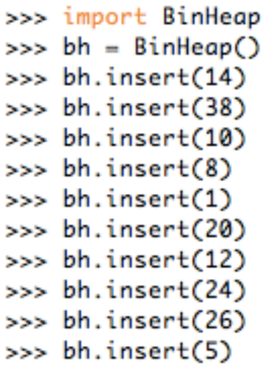
PS 6.11.5 Generate a random list of integers. Show the binary heap tree resulting from inserting the integers on the list one at a time.



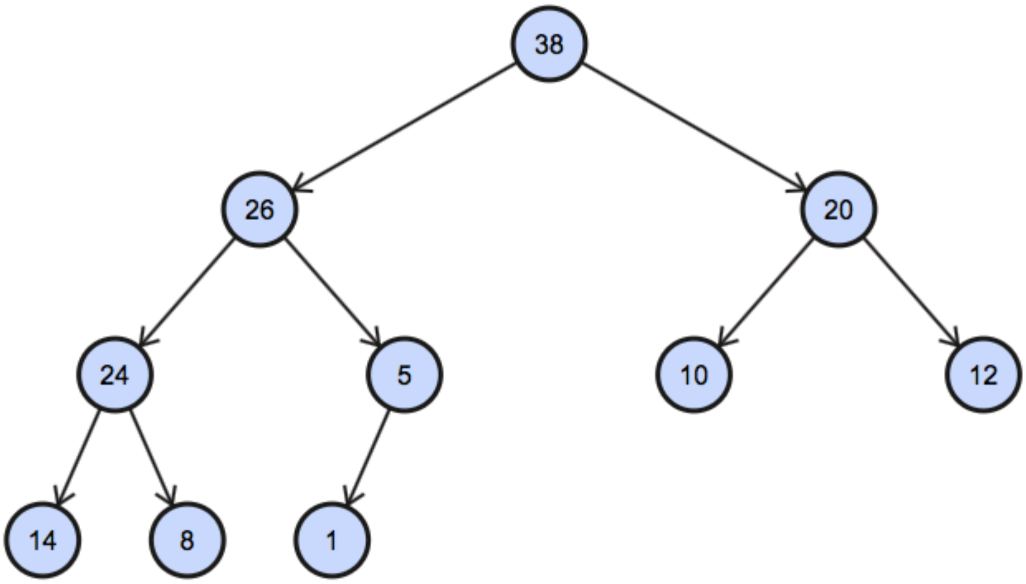
```
>>> import random #import random module
>>> random.sample(range(0, 50), 10) #generate a list of 10 numbers with in the range(0,50).
>>> [14, 38, 10, 8, 1, 20, 12, 24, 26, 5]
>>> class BinHeap:
        def __init__(self):
                self.heapList = [0]
                self.currentSize = 0

        def percUp(self,i):
                while i // 2 > 0:
                        if self.heapList[i] < self.heapList[i // 2]:
                                tmp = self.heapList[i // 2]
                                self.heapList[i // 2] = self.heapList[i]
                                self.heapList[i] = tmp
                        i = i // 2

        def percDown(self,i):
                while (i * 2) <= self.currentSize:
                        mc = self.minChild(i)
                        if self.heapList[i] >self.heapList[mc]:
                                tmp = self.heapList[i]
                                self.heapList[i] = self.heapList[mc]
                                self.heapList[mc] = tmp
                        i = mc

        def minChild(self,i):
                if i * 2 + 1 > self.currentSize:
                        return i * 2
                else:
                        if self.heapList[i*2] < self.heapList[i*2+1]:
                                return i * 2
                        else:
                                return i * 2 + 1
```



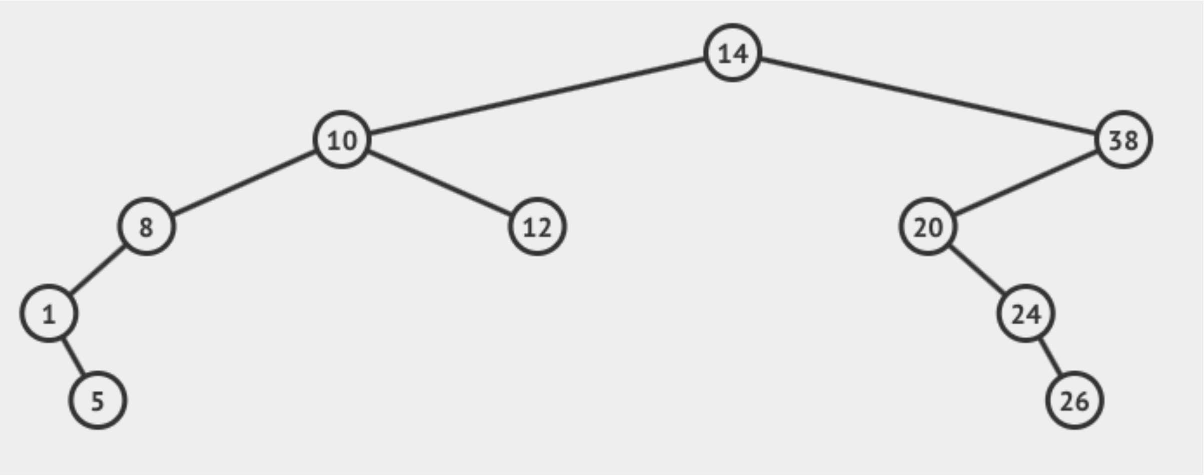
```
def insert(self,k):
        self.heapList.append(k)
        self.currentSize = self.currentSize + 1
        self.percUp(self.currentSize)


def buildHeap(self,alist):
        i = len(alist) // 2
        self.currentSize = len(alist)
        self.heapList = [0] + alist[:]
        while (i > 0):
                self.percDown(i)
                i = i -1
```



```
>>> import BinHeap
>>> bh = BinHeap()
>>> bh.insert(14)
>>> bh.insert(38)
>>> bh.insert(10)
>>> bh.insert(8)
>>> bh.insert(1)
>>> bh.insert(20)
>>> bh.insert(12)
>>> bh.insert(24)
>>> bh.insert(26)
>>> bh.insert(5)
```



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


14
10
38
8
12
20
1
24
5
26