# A7.py - program with functions to generate a string with three

# random lowercase letters, and returns a list of a specified size

# conforming to a regex string

# uses xeger library from here "https://github.com/crdoconnor/xeger"

# @author Erik Stryshak

# @sid 41069864

# @version 1.0

import random

import string

import re

from xeger import Xeger

class TripleGen(object):

def \_\_init\_\_(self, regex\_string, count):

self.regex\_string = regex\_string

self.count = count

def genRandomTriple(self):

# use random and string classes to generate three random

# lowercase letters and return a concatenated string

ch\_one = random.choice(string.ascii\_lowercase)

ch\_two = random.choice(string.ascii\_lowercase)

ch\_three = random.choice(string.ascii\_lowercase)

return ch\_one + ch\_two + ch\_three

def genFilteredTriple(self, regex\_string, count):

# using the Xeger library, generate "count" random strings

# that adhere to the regex string passed in

return\_list = []

for x in range(0, count):

str\_gen = Xeger(limit=10)

return\_list.append(str\_gen.xeger(regex\_string))

return return\_list

def getList(self):

# calls the genFilteredTriple method with this object's

# regex string and count

return self.genFilteredTriple(self.regex\_string, self.count)

class MyIterator(object):

def \_\_init\_\_(self, iterator\_list):

# constructor sorts the incoming list alphabetically

self.list = sorted(iterator\_list)

self.next\_pointer = 1

def next(self):

# use hasNext function to check if there is a next value

# if it does not exist, raise StopIteration

if self.hasNext():

self.next\_pointer += 1

return self.list[self.next\_pointer-1]

raise StopIteration

def hasNext(self):

# use size of the list and next\_pointer to determine if

# there is a next value in the list

if len(self.list) < self.next\_pointer + 1:

return False

return True

def main():

# create a TripleGen object

triple\_gen = TripleGen("[bcdfgmnpr][aeiou][dgnprstwxyz]", 4)

print ("Regex string used: [bcdfgmnpr][aeiou][dgnprstwxyz]")

print ("Generating 4 random triples results in:")

print (triple\_gen.getList()[0])

print (triple\_gen.getList()[1])

print (triple\_gen.getList()[2])

print (triple\_gen.getList()[3])

print ("\nFunction genRandomTriple examples:")

print (triple\_gen.genRandomTriple())

print (triple\_gen.genRandomTriple())

print (triple\_gen.genRandomTriple())

print (triple\_gen.genRandomTriple())

print ("\nIterater Demonstration")

my\_it = MyIterator(triple\_gen.getList())

# should return 2nd, 3rd, and 4th elements

print ("Has next = " + str(my\_it.hasNext()))

print ("Next element = " + my\_it.next())

print ("Has next = " + str(my\_it.hasNext()))

print ("Next element = " + my\_it.next())

print ("Has next = " + str(my\_it.hasNext()))

print ("Next element = " + my\_it.next())

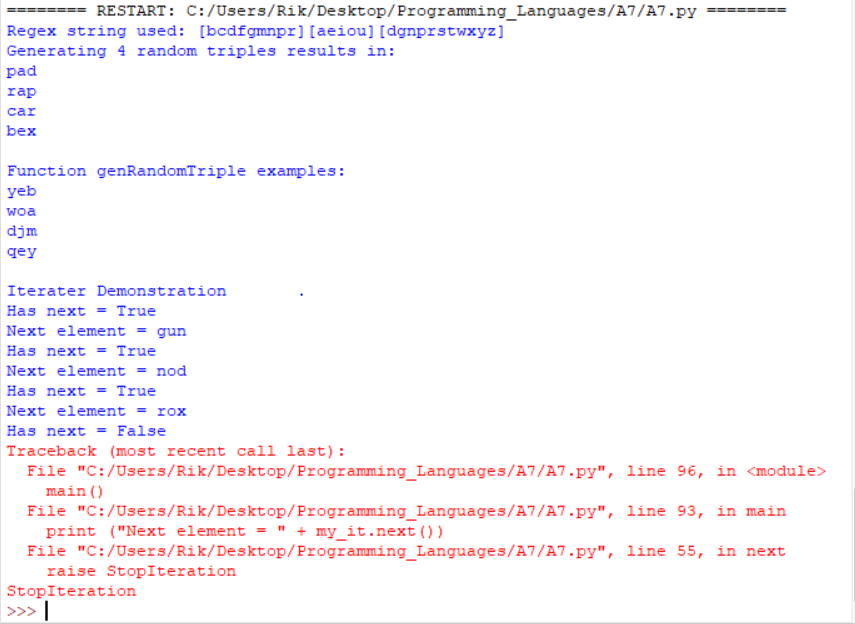
# should return false and throw StopIteration

print ("Has next = " + str(my\_it.hasNext()))

print ("Next element = " + my\_it.next())

if \_\_name\_\_ == "\_\_main\_\_":

main()



grammar triplegen;

stmt : TOKENONE VOWEL TOKENTWO

TOKENONE : [bcdfgmnpr]

VOWEL : [aeiou]

TOKENTWO : [dgnprstwxyz]