#!/usr/bin/python2.4 -tt

# Copyright 2010 Google Inc.

# Licensed under the Apache License, Version 2.0

# http://www.apache.org/licenses/LICENSE-2.0

# Google's Python Class

# http://code.google.com/edu/languages/google-python-class/

# Additional basic string exercises

# D. verbing

# Given a string, if its length is at least 3,

# add 'ing' to its end.

# Unless it already ends in 'ing', in which case

# add 'ly' instead.

# If the string length is less than 3, leave it unchanged.

# Return the resulting string.

def verbing(s):

if len(s) >= 3:

if s[-3:] == 'ing':

x = s + 'ly'

else:

x = s + 'ing'

else: return s

return x

# E. not\_bad

# Given a string, find the first appearance of the

# substring 'not' and 'bad'. If the 'bad' follows

# the 'not', replace the whole 'not'...'bad' substring

# with 'good'.

# Return the resulting string.

# So 'This dinner is not that bad!' yields:

# This dinner is good!

def not\_bad(s):

if s.find('not') < s.find('bad'):

x = s.replace(s[s.find('not'):s.find('bad') + 3], 'good')

else: return s

return x

# F. front\_back

# Consider dividing a string into two halves.

# If the length is even, the front and back halves are the same length.

# If the length is odd, we'll say that the extra char goes in the front half.

# e.g. 'abcde', the front half is 'abc', the back half 'de'.

# Given 2 strings, a and b, return a string of the form

# a-front + b-front + a-back + b-back

def front\_back(a, b):

ad = (len(a) + 1) // 2

bd = (len(b) + 1) // 2

return a[:ad] + b[:bd] + a[ad:] + b[bd:]

# x = len(a) / 2

# y = len(b) / 2

# if len(a) % 2 == 0 and len(b) % 2 == 0:

# return a[:x] + b[:y] + a[x:] + b[y:]

# elif len(a) % 2 != 0 and len(b) %2 != 0:

# return a[:x + 1] + b[:y + 1] + a[x + 1:] + b[y + 1:]

# elif len(a) % 2 == 0 and len(b) %2 !=0:

# return a[:x] + b[:y + 1] + a[x:] + b[y + 1:]

# elif len(a) %2 !=0 and len(b) %2 == 0:

# return a[:x + 1] + b[:y] + a[x + 1:] + b[y:]

# return

# Simple provided test() function used in main() to print

# what each function returns vs. what it's supposed to return.

def test(got, expected):

if got == expected:

prefix = ' OK '

else:

prefix = ' X '

print '%s got: %s expected: %s' % (prefix, repr(got), repr(expected))

# main() calls the above functions with interesting inputs,

# using the above test() to check if the result is correct or not.

def main():

print 'verbing'

test(verbing('hail'), 'hailing')

test(verbing('swiming'), 'swimingly')

test(verbing('do'), 'do')

print

print 'not\_bad'

test(not\_bad('This movie is not so bad'), 'This movie is good')

test(not\_bad('This dinner is not that bad!'), 'This dinner is good!')

test(not\_bad('This tea is not hot'), 'This tea is not hot')

test(not\_bad("It's bad yet not"), "It's bad yet not")

test(not\_bad('This dinner is not that that that that that that that bad! hey hey hey hey'), 'This dinner is good!')

print

print 'front\_back'

test(front\_back('abcd', 'xy'), 'abxcdy')

test(front\_back('abcde', 'xyz'), 'abcxydez')

test(front\_back('Kitten', 'Donut'), 'KitDontenut')

test(front\_back('Donut','Kitten' ), 'DonKitutten')

if \_\_name\_\_ == '\_\_main\_\_':

main()