# EditDistance.py

# Edit distance is a way to quantify the similarity of one string to another. It is measure as smallest the number of small "edits" one must make to one string, say s1, so that it exactly matches the other, say s2.

# Edits may take one of three forms:
# (1) Substitution: one can substitute one character for another in s1.
# (2) Insertion: one can add a character to s1.
# (3) Deletion: one can delete a character from s1.

# Eg. 1: if
# s1 = 'spam'
# s2 = 'poems'
# we substitute p for s, substitute o for p, substitute e for a, and insert an s at the end.
# => distance(s1, s2) = 4

# Eg. 2: if
# s1 = 'alien'
# s2 = 'sales'
# we add s to the start of s1, delete the e, and substitute a for the n.
# => distance(s1, s2) = 3

# Note: modifying s2 so that matches s1 requires the same number of edits. That is, the edit distance is symmetric.

# Following the textbook, let's develop our function in stages, doing the easy parts first

def distance(s1,s2):
    """The edit distance between s1 and s2.
    [The definition of edit distance might go here.]
    ""
    # if one of s1 or s2 is empty, it's easy
    if s1 == '':
        return len(s2) # Must add this number of chars to s1.
    elif s2 == '':
        return len(s1) # Must delete this number of chars from s1.
    else:
        return "???"
# Now, if both s1 and s2 are non-empty, we can compare their
# first characters:

def distance(s1, s2):
    """The edit distance between s1 and s2.
    [The definition of edit distance might go here.]
    """
    if s1 == '':
        return len(s2)
    elif s2 == '':
        return len(s1)
    elif s1[0] == s2[0]:
        # Well, we're fortunate here. We need only compute the edit
distance for the rest of the strings since the first chars
# match.
        return distance(s1[1:], s2[1:])
    else:
        # If the first characters DO NOT match, it's complicated.
        # Here, we could insert a character into s1, delete one,
# or change one. Whichever of these we choose, it should
# be the one resulting in the least (minimum) edit distance.
        return "???"

def distance(s1, s2):
    """The edit distance between s1 and s2.
    [The definition of edit distance might go here.]
    """
    if s1 == '':
        return len(s2)
    elif s2 == '':
        return len(s1)
    elif s1[0] == s2[0]:
        return distance(s1[1:], s2[1:])
    else:
        # We can substitute a character (the first in s2) for the
# first character in s1. At a cost of one edit, the first
# characters now match.
        substitution = 1 + distance(s1[1:], s2[1:])
        # We can try deleting the first character from s1.
# That costs one edit. Add that to the distance between the
# (shorter) s1 and s2.
        deletion = 1 + distance(s1[1:], s2)
        # We can try adding a character (the first character of s2)
# to s1. Costing one edit, the first characters now match.
        insertion = 1 + distance(s1, s2[1:])
        # Having tried all of these, which of these three edit
# distances do we choose?
        return min(substitution, deletion, insertion)

# Now, a couple of tests (why the mixed quotes???)

print "distance('spam', 'poems') =", distance('spam', 'poems')
print "distance('alien', 'sales') =", distance('alien', 'sales')

# What additional tests should we devise?