import re  
from django.shortcuts import render  
import math  
  
  
# Create your views here.  
  
from django.http import HttpResponse  
  
def index(request):  
 return HttpResponse("Hello this is it...")  
  
def encrypt(request):  
 key = "HACK"  
  
 def encryptMessage(msg):  
 cipher = ""  
  
 k\_indx = 0  
  
 msg\_len = float(len(msg))  
 msg\_lst = list(msg)  
 key\_lst = sorted(list(key))  
  
 col = len(key)  
  
 row = int(math.ceil(msg\_len / col))  
  
  
 fill\_null = int((row \* col) - msg\_len)  
 msg\_lst.extend('\_' \* fill\_null)  
  
  
 matrix = [msg\_lst[i: i + col]  
 for i in range(0, len(msg\_lst), col)]  
  
 for \_ in range(col):  
 curr\_idx = key.index(key\_lst[k\_indx])  
 cipher += ''.join([row[curr\_idx]  
 for row in matrix])  
 k\_indx += 1  
  
 return cipher  
  
  
 # Decryption  
 def decryptMessage(cipher):  
 msg = ""  
  
 k\_indx = 0  
  
 msg\_indx = 0  
 msg\_len = float(len(cipher))  
 msg\_lst = list(cipher)  
  
 col = len(key)  
  
 row = int(math.ceil(msg\_len / col))  
  
 key\_lst = sorted(list(key))  
  
  
 dec\_cipher = []  
 for \_ in range(row):  
 dec\_cipher += [[None] \* col]  
  
  
 for \_ in range(col):  
 curr\_idx = key.index(key\_lst[k\_indx])  
  
 for j in range(row):  
 dec\_cipher[j][curr\_idx] = msg\_lst[msg\_indx]  
 msg\_indx += 1  
 k\_indx += 1  
  
 try:  
 msg = ''.join(sum(dec\_cipher, []))  
 except TypeError:  
 raise TypeError("This program cannot",  
 "handle repeating words.")  
  
 null\_count = msg.count('\_')  
  
 if null\_count > 0:  
 return msg[: -null\_count]  
  
 return msg  
  
  
 print(" Welcome to ENCRYPTION USING TRANSPORTATION TECHNIQUE")  
 msg = str(request.GET['m'])  
 cipher = encryptMessage(msg)  
  
 en\_msg=format(cipher)  
  
 return render(request,'page.html',{'result\_message':en\_msg})   
  
  
  
def decryptMessage(request):  
  
 key = "HACK"  
  
 def encryptMessage(msg):  
 cipher = ""  
  
 k\_indx = 0  
  
 msg\_len = float(len(msg))  
 msg\_lst = list(msg)  
 key\_lst = sorted(list(key))  
  
 col = len(key)  
  
 row = int(math.ceil(msg\_len / col))  
  
  
 fill\_null = int((row \* col) - msg\_len)  
 msg\_lst.extend('\_' \* fill\_null)  
  
  
 matrix = [msg\_lst[i: i + col]  
 for i in range(0, len(msg\_lst), col)]  
  
 for \_ in range(col):  
 curr\_idx = key.index(key\_lst[k\_indx])  
 cipher += ''.join([row[curr\_idx]  
 for row in matrix])  
 k\_indx += 1  
  
 return cipher  
  
  
 # Decryption  
 def decryptMessage(cipher):  
 msg = ""  
  
 k\_indx = 0  
  
 msg\_indx = 0  
 msg\_len = float(len(cipher))  
 msg\_lst = list(cipher)  
  
 col = len(key)  
  
 row = int(math.ceil(msg\_len / col))  
  
 key\_lst = sorted(list(key))  
  
  
 dec\_cipher = []  
 for \_ in range(row):  
 dec\_cipher += [[None] \* col]  
  
  
 for \_ in range(col):  
 curr\_idx = key.index(key\_lst[k\_indx])  
  
 for j in range(row):  
 dec\_cipher[j][curr\_idx] = msg\_lst[msg\_indx]  
 msg\_indx += 1  
 k\_indx += 1  
  
 try:  
 msg = ''.join(sum(dec\_cipher, []))  
 except TypeError:  
 raise TypeError("This program cannot",  
 "handle repeating words.")  
  
 null\_count = msg.count('\_')  
  
 if null\_count > 0:  
 return msg[: -null\_count]  
  
 return msg  
  
  
   
 msg = str(request.GET['d'])  
 cipher = decryptMessage(msg)  
  
 de\_msg=format(cipher)  
  
 return render(request,'page.html',{'decryptedmessage':de\_msg})   
  
  
def encrypt1(request):  
   
  
# Python3 program to illustrate  
# Rail Fence Cipher Encryption  
# and Decryption  
  
# function to encrypt a message  
 def encryptRailFence(text, key):  
 # create the matrix to cipher  
 # plain text key = rows ,  
 # length(text) = columns  
 # filling the rail matrix  
 # to distinguish filled  
 # spaces from blank ones  
 rail = [['\n' for i in range(len(text))]  
 for j in range(key)]  
  
 # to find the direction  
 dir\_down = False  
 row, col = 0, 0  
  
 for i in range(len(text)):  
  
 # check the direction of flow  
 # reverse the direction if we've just  
 # filled the top or bottom rail  
 if (row == 0) or (row == key - 1):  
 dir\_down = not dir\_down  
  
 # fill the corresponding alphabet  
 rail[row][col] = text[i]  
 col += 1  
  
 # find the next row using  
 # direction flag  
 if dir\_down:  
 row += 1  
 else:  
 row -= 1  
 # now we can construct the cipher  
 # using the rail matrix  
 result = []  
 for i in range(key):  
 for j in range(len(text)):  
 if rail[i][j] != '\n':  
 result.append(rail[i][j])  
 return ("".join(result))  
  
  
 # This function receives cipher-text  
 # and key and returns the original  
 # text after decryption  
 def decryptRailFence(cipher, key):  
 # create the matrix to cipher  
 # plain text key = rows ,  
 # length(text) = columns  
 # filling the rail matrix to  
 # distinguish filled spaces  
 # from blank ones  
 rail = [['\n' for i in range(len(cipher))]  
 for j in range(key)]  
  
 # to find the direction  
 dir\_down = None  
 row, col = 0, 0  
  
 # mark the places with '\*'  
 for i in range(len(cipher)):  
 if row == 0:  
 dir\_down = True  
 if row == key - 1:  
 dir\_down = False  
  
 # place the marker  
 rail[row][col] = '\*'  
 col += 1  
  
 # find the next row  
 # using direction flag  
 if dir\_down:  
 row += 1  
 else:  
 row -= 1  
  
 # now we can construct the  
 # fill the rail matrix  
 index = 0  
 for i in range(key):  
 for j in range(len(cipher)):  
 if ((rail[i][j] == '\*') and  
 (index < len(cipher))):  
 rail[i][j] = cipher[index]  
 index += 1  
  
 # now read the matrix in  
 # zig-zag manner to construct  
 # the resultant text  
 result = []  
 row, col = 0, 0  
 for i in range(len(cipher)):  
  
 # check the direction of flow  
 if row == 0:  
 dir\_down = True  
 if row == key - 1:  
 dir\_down = False  
  
 # place the marker  
 if (rail[row][col] != '\*'):  
 result.append(rail[row][col])  
 col += 1  
  
 # find the next row using  
 # direction flag  
 if dir\_down:  
 row += 1  
 else:  
 row -= 1  
 return ("".join(result))  
  
  
 # Driver code  
   
 x=3  
   
 message = str(request.GET['m1'])  
 encrypted\_message =encryptRailFence(message,x)  
 return render(request,'page1.html',{'result\_message1':encrypted\_message })  
  
  
  
 # message2 = str(request.GET['m2'])  
 # decrypted\_message =decryptRailFence(message2,x)  
 # return render(request,'page1.html',{'decryptedmessage':decrypted\_message })  
   
  
 # msg = str(request.GET['m'])  
 # cipher = encryptMessage(msg)  
  
 # en\_msg=format(cipher)  
  
 # return render(request,'page.html',{'result\_message':en\_msg})  
  
def decrypt(request):  
 def decryptRailFence(cipher, key):  
 # create the matrix to cipher  
 # plain text key = rows ,  
 # length(text) = columns  
 # filling the rail matrix to  
 # distinguish filled spaces  
 # from blank ones  
 rail = [['\n' for i in range(len(cipher))]  
 for j in range(key)]  
  
 # to find the direction  
 dir\_down = None  
 row, col = 0, 0  
  
 # mark the places with '\*'  
 for i in range(len(cipher)):  
 if row == 0:  
 dir\_down = True  
 if row == key - 1:  
 dir\_down = False  
  
 # place the marker  
 rail[row][col] = '\*'  
 col += 1  
  
 # find the next row  
 # using direction flag  
 if dir\_down:  
 row += 1  
 else:  
 row -= 1  
  
 # now we can construct the  
 # fill the rail matrix  
 index = 0  
 for i in range(key):  
 for j in range(len(cipher)):  
 if ((rail[i][j] == '\*') and  
 (index < len(cipher))):  
 rail[i][j] = cipher[index]  
 index += 1  
  
 # now read the matrix in  
 # zig-zag manner to construct  
 # the resultant text  
 result = []  
 row, col = 0, 0  
 for i in range(len(cipher)):  
  
 # check the direction of flow  
 if row == 0:  
 dir\_down = True  
 if row == key - 1:  
 dir\_down = False  
  
 # place the marker  
 if (rail[row][col] != '\*'):  
 result.append(rail[row][col])  
 col += 1  
  
 # find the next row using  
 # direction flag  
 if dir\_down:  
 row += 1  
 else:  
 row -= 1  
 return ("".join(result))  
  
   
 x=3  
  
 message2 = str(request.GET['m2'])  
 decrypted\_message =decryptRailFence(message2,x)  
 return render(request,'page1.html',{'decryptedmessage':decrypted\_message })  
   
  
  
 # Driver code