```
# its on github
##### https://github.com/BeToast/forenzics/blob/main/ass1/playfair.py ######

import sys, re, string
import numpy as np

#########################  passed arg1 is string to encrypt, passed arg2 is key
#########################

if len(sys.argv)<2: # runs with defaults with no params passed
  print("less than two args passed. using defaults.\ntry passing two args like this : python3
playfair.py MESSAGE 'CIPHER KEY'\n")
  message = "MARY HAS A LITTLE LAMB ITS FLEECE AS WHITE AS SNOW"
  print(f"message : '{message}'")
  message = message.replace('0', 'O')
  key = "TOMATOJUICE"
  print(f"key : '{key}'")
  key = key.replace('0', 'O')

else: # if two params passed
  input_sus = False
  if re.search("^[A-Z0-9\s]+$", sys.argv[1]): # we allow zero's and replace them immediately
        message = sys.argv[1].replace('0', 'O') # replace
  else:
        print(f"your input '{sys.argv[1]}' violates the acceptable RegEx for this playfair
cipher\narguments must only contain captial letters, number 1-9, and space characters.")
        input_sus=True

  if re.search("^[A-Z0-9\s]+$", sys.argv[2]):
        key = sys.argv[2].replace('0', 'O')
  else:
        print(f"your input '{sys.argv[2]}' violates the acceptable RegEx for this playfair
cipher\narguments must only contain captial letters, number 1-9, and space characters.")
        input_sus=True

  if input_sus:
        sys.exit() # exit if input is not acceptable

#########################  cipher key setup  #########################

key_list = list(dict.fromkeys(key)) # use dist to remove duplicates, then store as list
#entire alphabet + 1-9 + space
alphabet_list = list(string.ascii_uppercase)+list(map(str, range(1,10)))+[' '] # create alphabet with
list concatination
```

```
cipher_list = list(dict.fromkeys(key_list+alphabet_list)) # concatenate key+alphabet then remove duplicates from end with dict.

cipher_grid = np.array(cipher_list).reshape(6,6) # reshape to 6x6, the grid is done here.
print(f"cipher_grid: \n{cipher_grid}")

#########################  message setup  #########################

message_pairs = []
i = 1
while i < len(message): # increment by two until end of string.
  curr_pair = [message[i-1],message[i]]
  if curr_pair[0] == curr_pair[1]: # if pair is duplicate, make seconds char X
        curr_pair[1] = 'X'
        i -= 1 # decrement so we dont skip that second letter we overwrote
  message_pairs.append(curr_pair)
  i += 2

  if i == len(message): # if only one char remaining
        message_pairs.append([message[i-1],'X'])
        break

print(f"message_pairs: \n{message_pairs}")

#########################  encrypt #########################

encrypted_pairs = []

for pair in message_pairs:
  char_one_coords = np.argwhere(cipher_grid == pair[0])[0]
  char_two_coords = np.argwhere(cipher_grid == pair[1])[0]
  if char_one_coords[0]==char_two_coords[0]: # same row, increment column index with modulo 6 to get excryption.
        encrypted_pairs.append([cipher_grid[char_one_coords[0]][(char_one_coords[1]+1)%6], cipher_grid[char_two_coords[0]][(char_two_coords[1]+1)%6]])
  elif char_one_coords[1]==char_two_coords[1]: # same col, increment row index with modulo 6 to get excryption.
        encrypted_pairs.append([cipher_grid[(char_one_coords[0]+1)%6][char_one_coords[1]], cipher_grid[(char_two_coords[0]+1)%6][char_two_coords[1]]])
  else: # otherwise, encryption is the origional coords, but the other coordinates column index.
        encrypted_pairs.append([cipher_grid[(char_one_coords[0])][char_two_coords[1]], cipher_grid[(char_two_coords[0])][char_one_coords[1]]])

print(f"\nencrypted_pairs: \n{encrypted_pairs}")
```