# Encoder for Larry, Decoder for Moe

# 1.py

# In 8.py (last lecture) we saw how Larry did encoding. Let's put that # logic in a function called encode(). Then we'll write another function # called decode() which Moe can use to extract the message.

# We'll put the dictionary d inside both functions, since d is supposed # to be private to both Larry and Moe

import sys

def search(list, item):
    # Look for the index of input char
    for i in range(0, len(list), 1):
        if (item == list[i]):
            return(i)

print("Error: Input character is not in the given alphabet")
sys.exit(0)  # always program defensively, so as not to be surprised

def encode(m, a):
    # encode msg m using alphabet a[] and dictionary [d]

d = ["f", "p", "s", "t", "o", "m", "z", "a", "d", "v", "x", "e", "l", "r", "w", "g", " ", "h", "j", "b", "i", "n", "u", "c", "y", "k", "q"]

coded_msg = []

for j in range(0, len(m), 1):
    index = search(a, m[j])  # get index in a
    coded_msg.append(d[index])  # get code char and append as we build

    cm = ".join(coded_msg)  # remove the "list-appearance" (quotes, commas),

    return(cm)

def decode(cm, a):
d = ["f", "p", "s", "t", "o", "m", "z", "a", "d", "v", "x", "e", "l", "r", "w", "g", "h", "j", "b", "i", "n", "u", "c", "y", "k", "q"]

decoded_msg = []

for j in range(0, len(cm), 1):
    index = search(d, cm[j])
    decoded_msg.append(a[index])  # just the opposite of what encode()

dm = ".".join(decoded_msg)

return(dm)

def main():
    # the alphabet is in a[]; no commas and apostrophes etc., and all message letters must be in lowercase

    a = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", " "]

    response = "Y"

    while (response == "Y"):
        m = input("What is Larry's message?:")

        c = encode(m, a)  # Larry encodes msg m
        print("Plaintext message: ", m)  # msg m
        print(" ")
        print("The coded message: ", c)  # msg m, encoded

        d = decode(c, a)  # Now Moe has to decode it

        print(" ")
        print("The decoded msg is: ", d)
        print(" __________________________________________ ")
        print(" ")
        response = input("Encode another message? Y/N: ")

        print ("Encoding/decoding is done")

# Note: This encoding can be cracked by studying patterns and trying to # reconstruct the dictionary. If Larry uses a random number seed to generate # a random dictionary and passes that seed to Moe secretly, then cracking # the code becomes more difficult.
# 2.py

# What is a Python sequence?
# It is a generic term for an ordered set. It helps you store things in an 
# ordered and thus efficient way.

# Python Sequences
#   I
#   I
#   I I I
# Lists Strings Tuples
#   ["B","i","l","b","o","5"]
#   ["i","m","m","u","t","a","b","l","e"]
# so a tuple is a list that cannot
# ever be changed

# All 3 types are Python OBJECTS
# All OBJECTS have METHODS (functions to do "object stuff")

# String methods: (see p134-p140)
# split(), join(), eval(),capitalize(),lower(),find(), count(), etc.

# List methods: (see p139-p141),(p345)
# append(), sort(), reverse(), insert(), etc.

# 3.py

# Let's convert a date that is input in mm/dd/yyyy form

import sys

def get_data():
    date = input("Please input the date (mm/dd/yyyy): ")
    mon, day, year = date.split("/")
# mon, day, year will be strings
# if there are leading zeros (i.e., 02/03) eval() will fail; use int()

mon = int(mon)
day = int(day)
year = int(year)

# now those strings are integers

# let's check for validity before returning values

if ((mon<=0) or (mon>12)):
    print("Bad month value")
sys.exit(0)

if ((day<=0) or (day>31)):
    print("Bad day value")
sys.exit(0)

if (year < 0):
    print("Bad year value")
sys.exit(0)

return(mon, day, year)

def main():

    # let's convert a date


    m, d, y = get_data()

    print("
    print("The date is:" ,months[m-1], d, "," , y)

    # Remember TYPE CONVERSION

    # float(<expr>) converts expr to floating point
    # int  (<expr>)    "        " integer
    # str  (<expr>)    "        " string
    # eval (<string>) evaluates string as an expression

#  #4.py
# All about FORMATS to control print output

def wait():
    x = input()

import math

def main():
    z = math.pi
    print("pi is ",z)
    wait()
    print(" ")
    print("Here it appears that Python prints 15 digits after the decimal point")

    wait()
    print(" ")

    print("Let's print just 5 digits after the decimal point")
    print(" ")

    print("pi is {0:0.5f}".format(z))

    wait()
    print(" ")

    print("Jack the math wiz earns $",z," every minute looks odd")

    print(" ")
    print("Jack the math wiz earns ${0:0.2f}".format(z)," every minute")

# Textbook uses {<index>:<format-specifier>}
# index is optional; when omitted, parameters go into slots from L to R
# Our example --> format-specifier = 0.5f
#                 <width>.<precision><type>

# width => how many spaces for value? (use " " padding if value needs less)
# less space allocated means value will use as much as needed
# 0. => 0 is not enough, so as much space as needed will be used
# 0.5 => precision is 5, rounded to 5 decimal places
# 0.5f => "f" is fixed point, so 5 places used anyway, even if all
0's

# Examples:

```python
wait()
s = "\{0\}, you \{1\}, your pay is \${2}\".format("Jack","Wiz",math.pi)
print(s)
wait()
s = "\{0\}, you \{1\}, your pay is \${2:.2f}\".format("Jack","Wiz",math.pi)
print(s)
wait()
s = "Int \{0:1\} put in field of width 1\".format(9)
print(s)
wait()
s = "Int \{0:15\} put in field of width 15\".format(9)
print(s)
wait()
s = "\{0:20.5\} has width 20 and precision 5\".format(z) #no f, rounding
print(s)
wait()
s = "\{0:20.5f\} has width 20 and precision 5\".format(z) #f, so 5 places
print(s)
wait()
s = "\{0:8.5f\} has width 8 and precision 5\".format(z) #f, so 5 places
print(s)
wait()
s = "\{0:0.5\} has width 0 and precision 5\".format(z) #no f, rounding
print(s)
```

# Now see what can happen with floating point numbers (approximations!)
```python
wait()
s = "Compare {0} and {0:.20}".format(3.14)
print(s)

# DEFAULTS: Strings => left-justified, Numeric values => right-justified

# How to change the default?

wait()
s = "Left justification: {0:<30}".format("Hey!")
print(s)
wait()

s = "Right justification: {0:>30}".format("Hey!")
print(s)
wait()

s = "Centered           : {0:^30}".format("Hey!")
print(s)
```

#5.py

# FILES (input and output). Really, just processing strings

# File = sequence of data in secondary memory (e.g., disk). It can contain
#       any data type, usually text.

# = (if it's text) a long string of text, or many text lines.

# End-of-line marker: special character, or sequence of characters

# Examples:

# \n means "line break"

#Bilbo
#Baggins
#
#take 5!

# On a file, this looks like

# Bilbo
Baggins

take 5

#Note: *Only when* string is printed does \n take effect. Not in string
import sys

def search(list, item): # Look for the index of input char
    for i in range(0, len(list), 1):
        if (item == list[i]):
            return (i) # in alphabet a
    print("Error: Input character is not in the given alphabet")
sys.exit(0) # always program defensively, so as not to be surprised

def encode(m, a): # encode msg m using alphabet a[] and dictionary [d]
    d = ["f", "p", "s", "t", "o", "m", "z", "a", "d", "v", "x", "e", "l", "r", "w", "g", "n", "h", "j", "b", "i", "n", "u", "c", "y", "k", "q"]
    coded_msg = []
    for j in range(0, len(m), 1):
        index = search(a, m[j]) # get index in a
        coded_msg.append(d[index]) # get code char and append as we build
    cm = ".".join(coded_msg) # remove the "list-appearance" (quotes, commas),
    return(cm)

def decode(cm, a):
    d = ["f", "p", "s", "t", "o", "m", "z", "a", "d", "v", "x", "e", "l", "r", "w", "g", "n", "h", "j", "b", "i", "n", "u", "c", "y", "k", "q"]
    decoded_msg = []
    for j in range(0, len(cm), 1):
        index = search(d, cm[j])
        decoded_msg.append(a[index]) #just the opposite of what encode() does
    dm = ".".join(decoded_msg)
    return(dm)
#Read everything in a file and print it out on screen

def main():
    fname = input("Enter filename: ")  # use any existing file. We'll use 1.py
    print(" ")
    infile = open(fname,"r")
    stuff = infile.read()  # means read remainder of file, maybe many lines
    print(stuff)
    infile.close()

#Note: the input() function does the same, reads everything, but discards \n
print(" ")
f = "data.txt"  # make sure you have this file; it must have at least 3 lines of data

    # all in lower case letters a-z
    infile = open(f,"r")
    for i in range(3):
        line = infile.readline()  #read a line until \n        print(line[:-1])  #slice, to get rid of \n        #or use end=""
    infile.close()

# Now let's encode it using Larry's encoder and write it on a file
# to pass to Moe

    a = ["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"," "]

print(" ")
f = "data.txt"
    infile = open(f,"r")

Moe = open("forMoe.txt","w")  #this file will contain encoded text for Moe

    for i in range(5):
        line = infile.readline()
        m = encode(line[:-1],a)  #don't pass \n to encoder
        print(m,file=Moe)