```
#       Fidelio is a toy program for teaching various encryption and decryption schemes.
#
#       Copyright 2009 Sam Kennerly
#
#    This file is part of Fidelio.
#
#    Fidelio is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    Fidelio is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with Fidelio.  If not, see <http://www.gnu.org/licenses/>.

from Fidelio_Functions import *

# --------------------- MAIN PROGRAM -----------------------------
# The main program handles input/output and calls functions.  The important code is in Fidelio_Functions.py .

selectmode = ""
selectcipher = ""

while selectmode.upper() != "Q" :
    print "-------------------------- "
    print "\nWhat would you like to do?"
    print "[E]ncrypt message\n[D]ecrypt message\n[C]rack encryption\n[G]enerate RSA keys\n[S]how alphabets"
    selectmode = raw_input("[Q]uit\n\n")

    # [E]ncrypt converts raw text (input by user) into a long or string using one of several schemes.
    if selectmode.upper() == "E" :
        encrypt_me = raw_input("\nEnter the message to be encrypted.\n")
        print "\nChoose an encycryption scheme:"
        selectcipher = raw_input("[S]ubstitution  [C]aesar  [D]odgson  [R]SA  \t\t")
        (alphabet, dictionary) = selectalphabet()
        # Unless the 96-char alphabet is used, all letters will be treated as capitals.
        if alphabet != ALPHABET96:
            encrypt_me = encrypt_me.upper()

        # Substitution encryption
        if selectcipher.upper() == "S" :
            cipherlist = makepackets(makenumbers(encrypt_me, dictionary), 4)
            print "\nNumerical message is:\n",
            for i in range(len(cipherlist)) :
                print str(cipherlist[i]).zfill(8),
            print

        # Caesar encryption
        elif selectcipher.upper() == "C" :
            try:
                shift = int(raw_input("\nHow far to shift?  [Press Enter for default of 3.] \t"))
            except ValueError :
                shift = int(3)
            print "\nCaesar cipher is:"
            print caesarshift(encrypt_me, alphabet, dictionary, shift), "\n"

        # Dodgson encryption
        elif selectcipher.upper() == "D" :
            password = str(raw_input("\nEnter a password.\n"))
            print "\nDodgson cipher is:"
            print dodgsonencrypt(encrypt_me, alphabet, dictionary, password)

        # RSA encryption
        elif selectcipher.upper() == "R" :
            print "Enter recipient's RSA number and public key separated by a space."
            RSApublicinfo = raw_input().split(None)
            rsan = long(RSApublicinfo[0])
            publickey = long(RSApublicinfo[1])
            cipherlist = RSAencrypt(makepackets(makenumbers(encrypt_me, dictionary), 4), rsan, publickey)
            print "\nRSA cipher is:"
            for i in range(len(cipherlist)) :
                print cipherlist[i],
            print


# [D]ecrypt converts encrypted string (input by user) into a text message.
    elif selectmode.upper() == "D" :
        decrypt_me = raw_input("\nEnter the message to be decrypted.\n")
        print "\nChoose a decryption scheme."
        selectcipher = raw_input("[S]ubstitution  [C]aesar  [D]odgson  [R]SA  \t\t")
        (alphabet, dictionary) = selectalphabet()

        # Substitution decryption
        if selectcipher.upper() == "S" :
            try :
                decrypt_list = [long(packet) for packet in decrypt_me.split()]
                print "\nDecrypted message:"
                print makeletters(unpack(decrypt_list, 4), alphabet)
            except TypeError:
                print "\nInput must be a number!\n"

        # Caesar decryption
        elif selectcipher.upper() == "C" :
            try:
                shift = int(raw_input("What is the Caesar shift value?  [Press Enter for default of 3.] \t"))
            except ValueError:
                shift = int(3)
            print "\nDecrypted message:"
            print caesarshift(decrypt_me, alphabet, dictionary, -1 * shift), "\n"

        # Dodgson decryption
        elif selectcipher.upper() == "D" :
            password = str(raw_input("\nEnter the password.\n"))
            print "\nDecrypted message:"
            print dodgsondecrypt(decrypt_me, alphabet, dictionary, password), "\n"

        # RSA decryption
        elif selectcipher.upper() == "R" :
            cipherlist = [long(packet) for packet in decrypt_me.split()]
            keylist = raw_input("Enter your RSA number and private key separated by a space. \n").split()
            rsan = long(keylist[0])
            privatekey = long(keylist[1])
            print "\nDecrypted message:"
            print makeletters(unpack(RSAdecrypt(cipherlist, rsan, privatekey), 4), alphabet)


# [C]rack attempts to break an adversary's encryption.
    elif selectmode.upper() == "C" :
        crack_me = raw_input("Enter the message to be cracked.\n\n")
        print "\nChoose a cracking method:"
        selectcipher = raw_input("[C]aesar shifts  [F]requency analysis \t")

        # Caesar crack: list all possible Caesar shifts and scan by eye
        if selectcipher.upper() == "C" :
            (alphabet, dictionary) = selectalphabet()
            print "\nAttempting all", len(alphabet), "nonzero Caesar shifts:\n"
            for i in range(len(alphabet)) :
                print i, caesarshift(crack_me, alphabet, dictionary, -1 * i)

        # Frequency analysis: list frequency with which each char appears
        elif selectcipher.upper() == "F" :
            results = freqanalyze(crack_me)
            print
            for letter, freq in results :
                print letter, freq


# [G]enerates public and private keys for a user based on a passphrase
    elif selectmode.upper() == "G" :
        # The file 10000primes.txt contains the first 10000 prime numbers to be stored as a list of strings.

        print "\nEnter two prime numbers separated by a space or press Enter for automatic key generation."
        twoprimes = raw_input().split(None)
        if twoprimes:
            prime0 = int(twoprimes[0])
            prime1 = int(twoprimes[1])
            totient = (prime0 - 1) * (prime1 - 1)
            rsan = prime0 * prime1
            print "Choose a number less than", totient, "coprime to", totient, "."
            publickey = long(raw_input())
            privatekey = findinverse(publickey, totient)
        else:
            RSAkeylist = generatekey()
            prime0 = RSAkeylist[0]
            prime1 = RSAkeylist[1]
            rsan = RSAkeylist[2]
            publickey = RSAkeylist[3]
            privatekey = RSAkeylist[4]
            totient = RSAkeylist[5]

        print "RSA number is", prime0, "*", prime1, "=", rsan, "and public key is", publickey, "."
        print "Totient is", totient, ".  Private key is", privatekey, "."
        print "Keep the totient, private key, and two prime factors secret!\n"
        print "Checking that public * private (mod totient) equals 1:"
        keychecker = ( publickey * privatekey ) % totient
        print publickey, "*", privatekey, "(mod", totient, ") = ", keychecker,
        if keychecker == 1 :
            print "  OK!"
        else :
            print "  Error!"


# "[S]how alphabet" prints an alphabet from Global Variables.
    elif selectmode.upper() == "S" :
        (printalphabet, dummy) = selectalphabet()           # the dummy variable doesn't do anything
        # Everything below this line is print formatting.
        numrows = int(len(printalphabet)/10 + 1)
        print
        for j in range(0, 10*numrows, 10):
            for i in range(10):
                if i+j < len(printalphabet):
                    print printalphabet[i+j], " ",
            print
            for i in range(10):
                if i+j < len(printalphabet):
                    print i+j,
                    if i+j < 10:
                        print " ",
                    elif i+j < 100:
                        print "",
            print "\n"

print
```