From: <http://rosettacode.org/wiki/RSA_code#Python>

Python[edit]

This example **may be incorrect** due to a recent change in the task requirements or a lack of testing. Please verify it and remove this message. If the example does not match the requirements or does not work, replace this message with Template:incorrect or fix the code yourself.

This code will open up a simple Tkinter window which has space to type a message. That message can then be encrypted by pressing the button labeled "encrypt". It will then print an output of ciphertext blocks, separated by commas. To decrypt a message, simply press the decrypt button. All ciphertext data must be entered with each block separated by commas. The ciphertext always goes (and appears) in the bottom box, while plaintext goes (and appears) in the topmost box. Upon decryption, random letters may have been appended to the end of the message, this is an aspect of the code to ensure the final block of plaintext is not a single letter, for example, a, 01, encoded is 01 (which means this letter was transmitted in the open!).

…[Note: The code for the Tkinter version has been removed for this document. (Tom Browder)]…

Note: the key given here is a toy key, it is easily broken.

Alternatively, a version without the tkinter window, which uses the same components as the program above, only without the Tkinter interface.

```
import random
import time

def decrypt(F,d):
    if d == 0:
        return 1
    if d == 1:
        return F
    w,r = divmod(d,2)
    if r == 1:
        return decrypt(F*F%n,w)*F%n
    else:
        return decrypt(F*F%n,w)

def correct():
    for i in range(len(C)):
        if len(str(P[i]))%2 !=0:
            y = str(0)+str(P[i])
            P.remove(str(P[i]))
            P.insert(i,y)

def cipher(b,e):
    if e == 0:
        return 1
    if e == 1:
        return b
    w,r = divmod(e,2)
```

```
    if r == 1:
        return cipher(b*b%n,w)*b%n
    else:
        return cipher(b*b%n,w)

def group(j,h,z):
    for i in range(int(j)):
        y = 0
        for n in range(h):
            y += int(numP[(h*i)+n])*(10**(z-2*n))
        X.append(int(y))

def gcd(a, b):
        while b != 0:
            (a, b) = (b, a%b)
        return a



letter =
["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",",",".","!","?"," "]
number = ["01","02","03","04","05","06","07","08","09","10","11","12","13",

"14","15","16","17","18","19","20","21","22","23","24","25","26","27",
          "28","29","30","31"]


print( '\n' )
def Decrypt():
    #decrypts an encoded message
    global m,P,C,x,h,p,Text,y,w
    P = []
    C = str(input("Enter ciphertext blocks:"))
    C = C.lstrip('[')
    C = C.rstrip(']')
    C = C.split(',')
    for i in range(len(C)):
        x = decrypt(int(C[i]),d)
        P.append(str(x))
    correct()
    #print(P)
    h = len(P[0])
    p = []
    for i in range(len(C)):
        for n in range(int(h/2)):
            p.append(str(P[i][(2*n):((2*n)+2)]))

    Text = []
    for i in range(len(p)):
        for j in range(len(letter)):
            if str(p[i]) == number[j]:
                Text.append(letter[j])
    PText = str()
    for i in range(len(Text)):
        PText = PText + str(Text[i])
```

```
    print("Plaintext is:", PText)

def Encrypt():
    #encrypts a plaintext message using the current key
    global plaintext,numP,q,j,z,X,C
    plaintext =(input("Enter Plaintext :"))
    plaintext = plaintext.lower()
    numP = []
    for i in range(len(plaintext)):
        for j in range(len(letter)):
            if plaintext[i] == letter[j]:
                numP.append(number[j])
    h = (len(str(n))//2)-1
    q = len(numP)%h
    for i in range(h-q):
        numP.append(number[random.randint(0,25)])
    j = len(numP) / h
    #print(numP)
    X = []
    z = 0
    for m in range(h-1):
        z+=2
    group(j,h,z)
    k = len(X)
    C = []
    for i in range(k):
        b = X[i]
        r = cipher(b,e)
        C.append(r)
    print("Ciphertext:",C)
    print("Number of Ciphertext blocks:",len(C))

def setup():
    global n,e,d
    while True:
        try:
            n = int(input(" Enter a value for n :"))
            if n > 2:
                break
        except ValueError:
            print('please enter a number')
    while 1!=2 :
        try:
            e = int(input(" Enter a value for e :"))
            if e >= 2:
                break
        except ValueError:
            print('please enter a number')
    while True:
        try:
            d = int(input(" Enter a value for d. If d unknown, enter 0 :"))
            if d >= 0:
                break
        except ValueError:
            print('please enter a number')
```

```
#setup()
n = 2537
e = 13
d = 937

print("To redefine n,e, or d, type 'n','e',... etc.")
print("To encrypt a message with the current key, type 'Encrypt'")
print("To decrypt a message with the current key, type 'Decrypt'")
print("Type quit to exit")
print( '\n' )
print( '\n' )

mm = str()
while mm != 'quit':
    mm = input("Enter Command...")
    if mm.lower() == 'encrypt':
        Encrypt()
    elif mm.lower() == 'decrypt':
        Decrypt()
    elif mm.lower() == 'n':
        try:
            print('current n = ',n)
            n = int(input(" Enter a value for n :"))
        except ValueError:
            print('That is not a valid entry')
    elif mm.lower() == 'help':
        print("To redefine n,e, or d, type 'n','e',... etc.")
        print("To encrypt a message with the current key, type 'Encrypt'")
        print("To decrypt a message with the current key, type 'Decrypt'")
        print("Type quit to exit")
        print( '\n' )
        print( '\n' )
    elif mm.lower() == 'e':
        try:
            print('current e = ',e)
            e = int(input(" Enter a value for e :"))
        except ValueError:
            print('That is not a valid entry')
    elif mm.lower() == 'd':
        try:
            print('current d = ',d)
            d = int(input(" Enter a value for d :"))
        except ValueError:
            print('That is not a valid entry')
    else:
        if mm != 'quit':
            ii= random.randint(0,6)
            statements = ["I sorry, Dave. I'm afraid i can't do that","I'm
begging you....read the directions","Nah ahh ahh, didnt say the magic
word","This input is....UNACCEPTABLE!!","Seriously....was that even a
word???","Please follow the directions","Just type 'help' if you are really
that lost"]
            print(statements[ii])
```

Example use :

Any entered commands are **not** case sensitive, nor is the plaintext input. Commands must be spelled correctly. Ciphertext blocks are input all at once, but must be separated by commas. When decrypted, there may be random letters attached to the end of the message. The program does this in order to fill blocks completely, and not have orphaned characters.

```
>>>
To redefine n,e, or d, type 'n','e',... etc.
To encrypt a message with the current key, type 'Encrypt'
To decrypt a message with the current key, type 'Decrypt'
Type quit to exit



Enter Command...ENCRYPT
Enter Plaintext :drink MORE Ovaltine
Ciphertext: [140, 2222, 1864, 1616, 821, 384, 2038, 2116, 2222, 205, 384,
2116, 45, 1, 2497, 793, 1864, 1616, 205, 41]
Number of Ciphertext blocks: 20
Enter Command...decrypt
Enter ciphertext blocks:[140, 2222, 1864, 1616, 821, 384, 2038, 2116, 2222,
205, 384, 2116, 45, 1, 2497, 793, 1864, 1616, 205, 41]
Plaintext is: drink more ovaltineu
Enter Command...quit
>>>
```