//Part A Miller- Rabin Algorithm

import random

def power(x, y, p):

    res = 1;

    x = x % p;

    while (y > 0):

        if (y & 1):

            res = (res \* x) % p;

        y = y>>1; # y = y/2

        x = (x \* x) % p;

    return res;

def miillerTest(d, n):

    a = 2 + random.randint(1, n - 4);

    # Compute a^d % n

    x = power(a, d, n);

    if (x == 1 or x == n - 1):

        return True;

    while (d != n - 1):

        x = (x \* x) % n;

        d \*= 2;

        if (x == 1):

            return False;

        if (x == n - 1):

            return True;

    # Return composite

    return False;

def isPrime( n, k):

    if (n <= 1 or n == 4):

        return False;

    if (n <= 3):

        return True;

    d = n - 1;

    while (d % 2 == 0):

        d //= 2;

    # Iterate given nber of 'k' times

    for i in range(k):

        if (miillerTest(d, n) == False):

            return False;

    return True;

k = 4;

print("All primes smaller than 100: ");

for n in range(1,100):

    if (isPrime(n, k)):

        print(n , end=" ");

//Part B AES Algorithm

sbox = [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,

        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,

        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,

        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,

        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,

        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,

        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,

        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,

        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,

        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,

        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,

        0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,

        0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,

        0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,

        0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,

        0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16]

Rcon = [0x00000000, 0x01000000, 0x02000000,

        0x04000000, 0x08000000, 0x10000000,

        0x20000000, 0x40000000, 0x80000000,

        0x1b000000, 0x36000000]

def keyExpansion(key):

    #prep w list to hold 44 tuples

    w = [()]\*44

    #fill out first 4 words based on the key

    for i in range(4):

        w[i] = (key[4\*i], key[4\*i+1], key[4\*i+2], key[4\*i+3])

    #fill out the rest based on previews words, rotword, subword and rcon values

    for i in range(4, 44):

        #get required previous keywords

        temp = w[i-1]

        word = w[i-4]

        #if multiple of 4 use rot, sub, rcon etc

        if i % 4 == 0:

            x = RotWord(temp)

            y = SubWord(x)

            rcon = Rcon[int(i/4)]

            temp = hexor(y, hex(rcon)[2:])

        #creating strings of hex rather than tuple

        word = ''.join(word)

        temp = ''.join(temp)

        #xor the two hex values

        xord = hexor(word, temp)

        w[i] = (xord[:2], xord[2:4], xord[4:6], xord[6:8])

    return w

#takes two hex values and calculates hex1 xor hex2

def hexor(hex1, hex2):

    #convert to binary

    bin1 = hex2binary(hex1)

    bin2 = hex2binary(hex2)

    #calculate

    xord = int(bin1, 2) ^ int(bin2, 2)

    #cut prefix

    hexed = hex(xord)[2:]

    #leading 0s get cut above, if not length 8 add a leading 0

    if len(hexed) != 8:

        hexed = '0' + hexed

    return hexed

#takes a hex value and returns binary

def hex2binary(hex):

    return bin(int(str(hex), 16))

#takes from 1 to the end, adds on from the start to 1

def RotWord(word):

    return word[1:] + word[:1]

#selects correct value from sbox based on the current word

def SubWord(word):

    sWord = ()

    #loop throug the current word

    for i in range(4):

        #check first char, if its a letter(a-f) get corresponding decimal

        #otherwise just take the value and add 1

        if word[i][0].isdigit() == False:

            row = ord(word[i][0]) - 86

        else:

            row = int(word[i][0])+1

        #repeat above for the seoncd char

        if word[i][1].isdigit() == False:

            col = ord(word[i][1]) - 86

        else:

            col = int(word[i][1])+1

        #get the index base on row and col (16x16 grid)

        sBoxIndex = (row\*16) - (17-col)

        #get the value from sbox without prefix

        piece = hex(sbox[sBoxIndex])[2:]

        #check length to ensure leading 0s are not forgotton

        if len(piece) != 2:

            piece = '0' + piece

        #form tuple

        sWord = (\*sWord, piece)

    #return string

    return ''.join(sWord)

#used to display the keywords neatly in this form: w0 = 0f 15 71 c9

def prettyPrint(w):

    print("\n\nKeywords: \n")

    for i in range(len(w)):

        print("w" + str(i), "=", w[i][0], w[i][1], w[i][2], w[i][3])

def main():

    #hardcoding input key for demonstration purposes, could be read in from user/program via cmd/gui etc.

    key = ["0f", "15", "71", "c9", "47", "d9", "e8", "59", "0c", "b7", "ad", "d6", "af", "7f", "67", "98"]

    #expand key

    w = keyExpansion(key)

    #display nicely

    print("Key provided: " + "".join(key))

    prettyPrint(w)

if \_\_name\_\_ == '\_\_main\_\_':

    main()