Question 5)

Implement a Sage function to perform e ncryption/decryption with an mxm Hill Cipher. The key should be an invertible Sage matrix over the integers mod 26. Do not just call the built in sage functionality for the Hill cipher. Show the operation of your function on the functions you write on a plaintext of your choice. You may use any functions you wrote for the previous question to answer this question.

Question 5)

Implement a Sage function to perform e ncryption/decryption with an mxm Hill Cipher. The key should be an invertible Sage matrix over the integers mod 26. Do not just call the built in sage functionality for the Hill cipher. Show the operation of your function on the functions you write on a plaintext of your choice. You may use any functions you wrote for the previous question to answer this question.

def validate\_hill\_key(K):

m = K.nrows()

if (K not in MatrixSpace(IntegerModRing(26), m)):

raise ValueError, "Key must be square matrix over the integers mod 26."

if (not K.is\_invertible()):

raise ValueError, "Key must be an invertible matrix."

return m

def string\_to\_hill\_block(in\_string, m):

instr\_list = [char\_to\_num(in\_string[j]) for j in xrange(m)]

FM = FreeModule(R, m)

return FM(instr\_list)

def hill\_block\_to\_string(in\_block, m):

outtxt = ""

for j in xrange(m):

outtxt += num\_to\_char(in\_block[j])

return outtxt

def hill\_process\_text(intext, M, m):

# strip out all nonalphabetic characters

intxt\_alpha\_only = ""

for j in xrange(len(intext)):

c = intext[j]

if is\_alphabetic\_char(c):

intxt\_alpha\_only += c

# pad to make length a multiple of m

len\_rem\_m = len(intxt\_alpha\_only) % m

if (0 != len\_rem\_m):

for j in xrange(m - len\_rem\_m):

intxt\_alpha\_only += 'a'

outtext = ""

num\_blocks = len(intxt\_alpha\_only) / m

for j in xrange(num\_blocks):

this\_block\_str = intxt\_alpha\_only[j\*m:(j+1)\*m]

vec = string\_to\_hill\_block(this\_block\_str, m)

out\_vec = M\*vec

outtext += hill\_block\_to\_string(out\_vec, m)

return outtext

def hill\_encrypt(plaintext, K):

m = validate\_hill\_key(K)

return hill\_process\_text(plaintext, K, m)

def hill\_decrypt(ciphertext, K):

m = validate\_hill\_key(K)

Kinv = K.inverse()

return hill\_process\_text(ciphertext, Kinv, m)

The following shows the operation of these functions on a plaintext:

sage: K = random\_matrix(R, 7)

sage: while (not K.is\_invertible()):

....: K = random\_matrix(R, 7)

....:

sage: K

[11 17 17 12 12 20 3]

[ 7 6 5 2 18 17 3]

[ 6 24 9 13 19 18 11]

[ 5 15 22 24 9 17 1]

[18 3 4 1 25 3 12]

[ 4 18 13 2 3 14 11]

[ 3 3 12 18 15 24 19]

sage: pt = "billyalsolikespurpledinosaurs"

sage: ct = hill\_encrypt(pt, K)

sage: ct

'hzdaxmqllpamnyzrkfinnshtgbpgqwemmuc'

sage: hill\_decrypt(ct, K)

'billyalsolikespurpledinosaursaaaaaa'