Question 3

Either using the functions you wrote in the previous question or the built in polynomial arithmetic in sage, as well as, either the given polynomial extended gcd, or the built in Sage extended gcd, implement a four function calculator for GF(24) with modulus x4+x+1. Consider elements of GF(24) to be degree 4 (fixed precision) polynomials in the primitive element. i.e., GF(24) elements are represented by lists of 4 binary values. You may use the underlying polynomial functions in sage, or any functions you wrote for the previous questions.

1. addition
2. scalar multiplication
3. multiplication
4. inversion

Question 3

Either using the functions you wrote in the previous question or the built in polynomial arithmetic in sage, as well as, either the given polynomial extended gcd, or the built in Sage extended gcd, implement a four function calculator for GF(24) with modulus x4+x+1. Consider elements of GF(24) to be degree 4 (fixed precision) polynomials in the primitive element. i.e., GF(24) elements are represented by lists of 4 binary values. You may use the underlying polynomial functions in sage, or any functions you wrote for the previous questions.

1. addition

gf2\_poly\_ring.<x> = GF(2)[]

gf16\_modulus = x^4 + x + 1

# solution to part a

def gf16\_add(f, g):

h = [f[j] + g[j] for j in xrange(4)]

return h

1. scalar multiplication

def gf16\_scalar\_multiply(f, c):

h = [c\*f[j] for j in xrange(4)]

return h

1. multiplication

def gf16\_multiply(f, g):

f\_poly = gf2\_poly\_ring(f)

g\_poly = gf2\_poly\_ring(g)

(q,h) = (f\*g).quo\_rem(gf16\_modulus)

return h.list()

1. inversion

def gf16\_inversion(f):

gf2\_poly\_ring(f)

(g, h, foo) = f.quo\_rem(gf16\_modulus)

return h.list()