Solutions 2

```
#Solution 1
def mycoprime(N):
    return [x for x in xrange(N) if gcd(x,N) == 1]
```

```
mycoprime(100)
```

```
[1, 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39, 41
47, 49, 51, 53, 57, 59, 61, 63, 67, 69, 71, 73, 77, 79, 81, 83,
89, 91, 93, 97, 99]
```

```
#Solution 2.1
def GoldbachConjecture(N):
    if (N%2 != 0 or N < 4):
        print ("N should be even and bigger than 2")
    else:
        return exists(xrange(2,N/2+2), lambda x:is_prime(x) and
is_prime(N-x))[0]
```

```
GoldbachConjecture(100)
```

```
True
```

```
#Solution 2.2
def GoldbachList(N):
    if (N%2 != 0 or N < 4):
        print ("N should be even and bigger than 2")
    else:
        return map(lambda n:[n,N-n], filter(lambda x :
is_prime(x) and is_prime(N-x), range(2,N/2+1)))
```

```
GoldbachList(100)
```

```
[[3, 97], [11, 89], [17, 83], [29, 71], [41, 59], [47, 53]]
```

```
#Solution 3
def groups(N):
    F = Integers(N)
    if F.multiplicative_group_is_cyclic() :
        print "The group is cyclic"
    else:
        print "The group is not cyclic"
    gs = F.unit_gens()
```

```
    print gs
    #Now, we try all possible combinations of generators
    G = [1]
    for x in gs:
        new = []
        for i in xrange(1,x.multiplicative_order()):
            for a in G:
                new.append(x^i*a)
        G.extend(new)
    G.sort()
    print G
    x = G[randint(0,len(G)-1)]
    print "Lagrange's theorem is " + str(len(G) %
x.multiplicative_order() == 0)
```

```
groups(88)
```

```
    The group is not cyclic
    [23, 45, 57]
    [1, 3, 5, 7, 9, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 35, 37,
    41, 43, 45, 47, 49, 51, 53, 57, 59, 61, 63, 65, 67, 69, 71, 73,
    79, 81, 83, 85, 87]
    Lagrange's theorem is True
```

```
#Solution 4
#The trick for this exercise is to use binary search to find a
possible base.

#Given n and b, return x such that x^b = n (if such an x exists)
using binary search
def binarySearch(n,b):
    l = 1
    u = 2^(floor(log(n,2)/b) + 1) #Maximum possible base
    while(l <=u):
        x = floor((l+u)/2)
        y = x^b
        if y == n:
            return x
        elif y<n:
            l = x+1
        else:
            u = x-1
    return false
```

```
#Solution 4 (cont)
```

```
#returns (x,b) s.t. x^b = n
def perfectPower(n):
    if n == 1:
        return (1,2)

    #We test for 2
    val = binarySearch(n,2)
    if val:
        return (val,2)

    b = 3 # The possible exponent
    upperbound = ceil(log(n,2))
    while b <= upperbound: #For every possible exponent, we try
to find a base
        if is_prime(b) : #We need to test only prime b's
            val = binarySearch(n,b)
            if val:
                return (val,b)
        b += 2
    return false
```

```
print perfectPower(589700950065322297792301221688 23)
print perfectPower(123^1237)
print perfectPower(456456456)
```

```
(34567, 7)
(123, 1237)
False
```