Modular operations addition, multiplication, congruence mod m

In [1]:
```
# modular addition/multiplication (trivial)
def addMod(a,b,m) :
    return (a + b)%m

def multMod(a,b,m) :
    return (a*b)%m
```

In [8]:
```
multMod(34,21,11)
```

Out[8]: 10

In [9]:
```
# congruence mod m
def congMod(a,b,m) :
    return (a-b)%m == 0
```

In [12]:
```
congMod(12,27,4)
```

Out[12]: False

Exponentiation mod m

1. Recursively (to demonstrate the principle)
2. Principal iterative version with exponential speed-up
3. How it's actually used

In [13]:
```
# Just for demonstration - there are better ways to compute that
def expModRec(a,n,m) :
    if n == 0 : return 1
    else      : return (a%m)*expModRec(a,n-1,m) % m
```

In [14]:
```
expModRec(123,456,987)
```

Out[14]: 267

Fast (?) algorithm for exponentiation mod m - (x^e) mod m
Fast if the standard exponentiation algorithm doesn't use binary templates

In [5]:
```
def fastExpModManual(x,e,m):
     X = x
     E = e
     Y = 1
     while E > 0:
         if E % 2 == 0:         # Even - divide by two for exponentia
l speedup
             X = (X * X) % m
             E = E/2
         else:
             Y = (X * Y) % m    # Odd - subtract one and then jump ne
xt round
             E = E - 1
     return Y
```

In [6]:
```
fastExpModManual(123,456,987)
```

Out[6]: 267

The Python pow method has binary speed-up

That's what we're going to use

In [15]:
```
def fastExpMod(x, e, m) :    # Just to make code "portable"
    return pow(x,e,m)        # Well, this thing uses binary templates
and is A LOT faster!
```

In [16]:
```
fastExpMod(123,456,987)
```

Out[16]: 267

Use the Extended Euclidean Algorithm to compute the first Bézout coefficient

In [19]:
```
from modutils import xgcd
def modInverse(a,m) :
    _, s, _ = xgcd(a,m)
    return s%m
```

In [20]:
```
modInverse(3,4)
```

Out[20]: 3

Now you can use the modular inverse if it exists, to divide mod a prime number

In [21]:
```
# Compute (a/b) mod p for prime p
# We're not testing that
# USE AT YOUR OWN RISK
def divMod(a,b,p) :
    return multMod(a,modInverse(b,p),p)
```

In [23]:
```
divMod(12,5,17)
```

Out[23]:
```
16
```