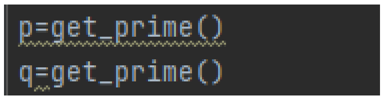
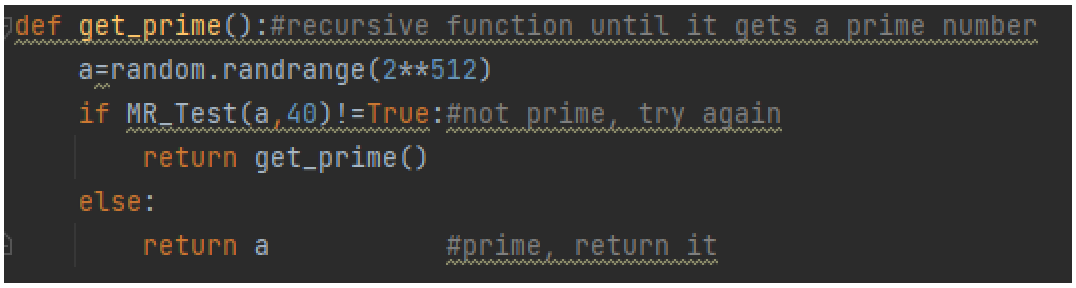
The first thing I did in this project was trying to implement and testing out individual functions to see if they work.

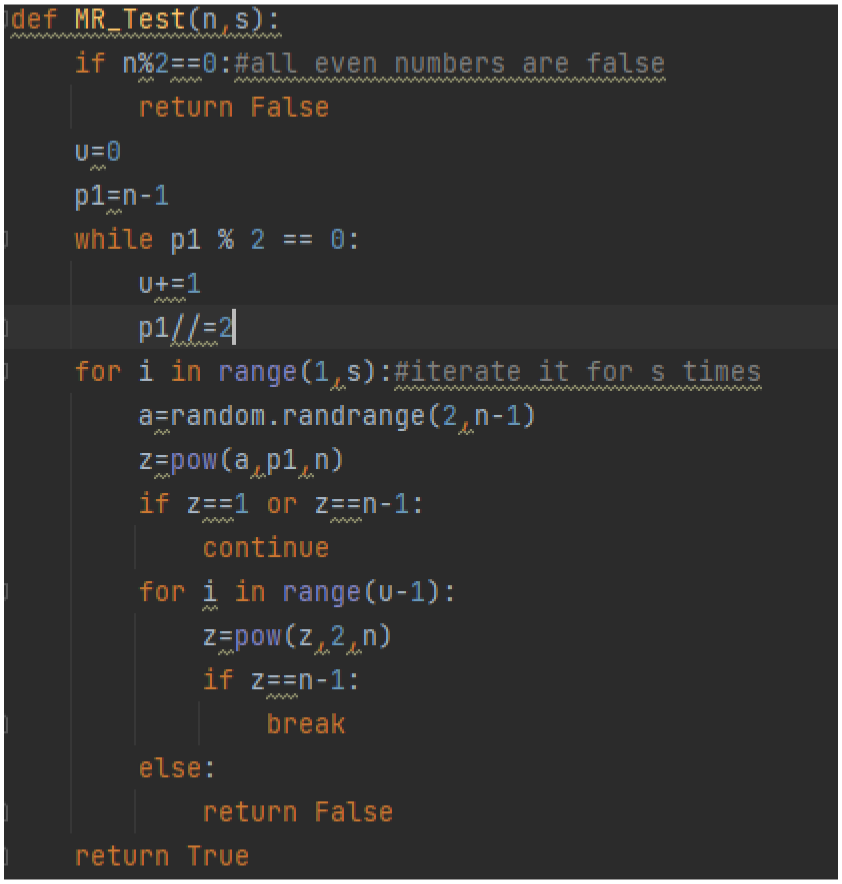
1.Code for **Miller-Rabin primality test** with s time iteration of 40 to make sure along with the code to just get a prime number of 512bits.



```
p=get_prime()
q=get_prime()
```



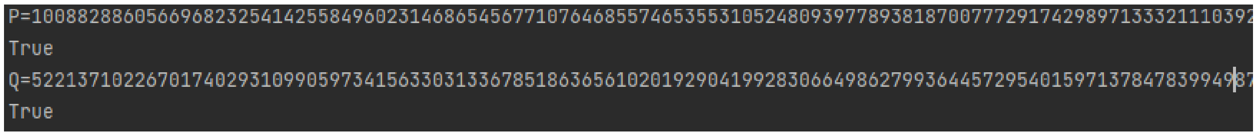
```
def get_prime():#recursive function until it gets a prime number
    a=random.randrange(2**512)
    if MR_Test(a,40)!=True:#not prime, try again
        return get_prime()
    else:
        return a            #prime, return it
```



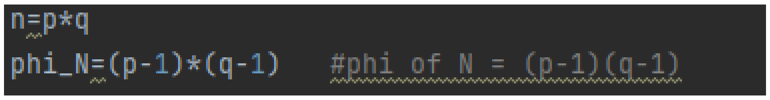
```
def MR_Test(n,s):
    if n%2==0:#all even numbers are false
        return False
    u=0
    p1=n-1
    while p1 % 2 == 0:
        u+=1
        p1//=2
    for i in range(1,s):#iterate it for s times
        a=random.randrange(2,n-1)
        z=pow(a,p1,n)
        if z==1 or z==n-1:
            continue
        for i in range(u-1):
            z=pow(z,2,n)
            if z==n-1:
                break
        else:
            return False
    return True
```

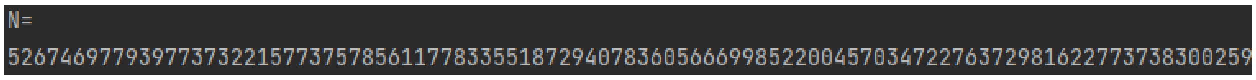
Getting and printing out P and Q, this is only a segment of it, there are too many numbers to fit them all into the screen. The True print out was to make sure they were prime. With the

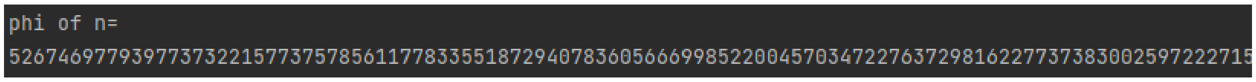
Miller-Rabin Primality Test. Each are size of s=512 bits long


```
P=100882886056696823254142558496023146865456771076468557465355310524809397789381870077729174298971333211103 92
True
Q=5221371022670174029310990597341563303133678518636561020192904199283066498627993644572954015971378478399 4987
True
```

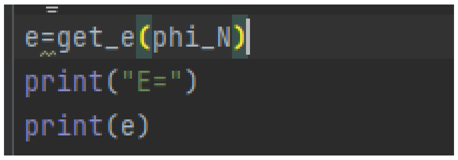
2.Getting n=p*q and Phi of n

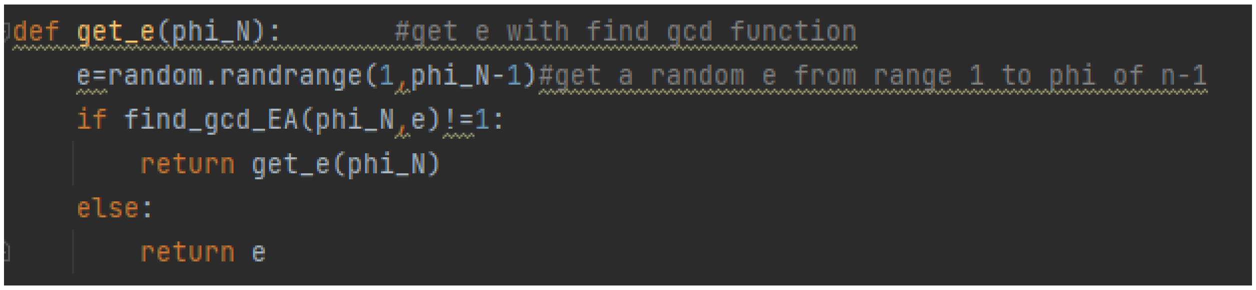

```
n=p*q
phi_N=(p-1)*(q-1)    #phi of N = (p-1)(q-1)
```

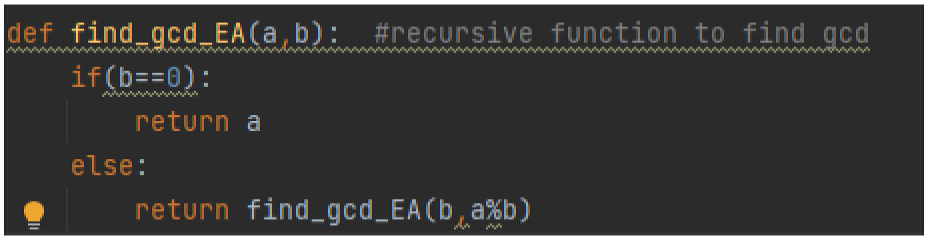

```
N=
52674697793977373221577375785611778335518729407836056669985220045703472276372981622773738300259
```

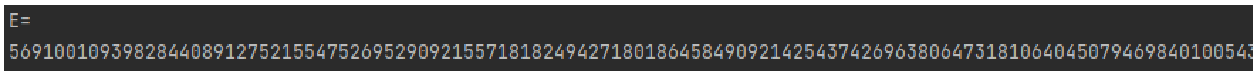

```
phi of n=
526746977939773732215773757856117783355187294078360566699852200457034722763729816227737383002597222715
```

3.Getting a random e from range 1 to phi(n)-1 with EA to check if gcd is = 1

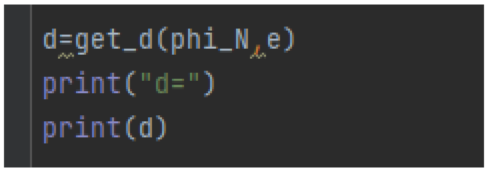

```
e=get_e(phi_N)
print("E=")
print(e)
```


```
def get_e(phi_N):        #get e with find gcd function
    e=random.randrange(1,phi_N-1)#get a random e from range 1 to phi of n-1
    if find_gcd_EA(phi_N,e)!=1:
        return get_e(phi_N)
    else:
        return e
```


```
def find_gcd_EA(a,b):  #recursive function to find gcd
    if(b==0):
        return a
    else:
        return find_gcd_EA(b,a%b)
```

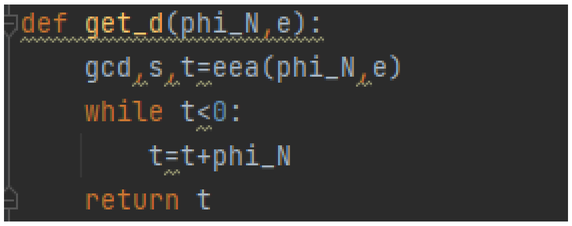

```
E=
56910010939828440891275215547526952909215571818249427180186458490921425437426963806473181064045079469840100543
```

4.Getting d after I have gotten phi(n) and e and using EEA to find inverse t.

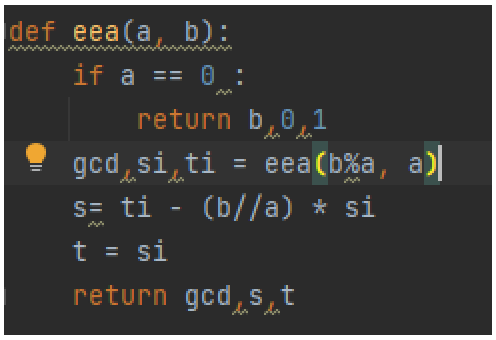


```
d=get_d(phi_N,e)
print("d=")
print(d)
```

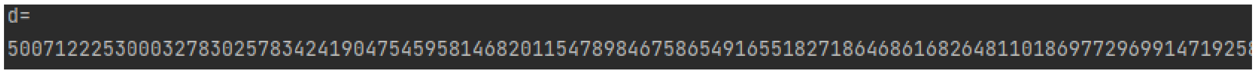
Here if t was negative, I add phi(n) until it becomes positive



```
def get_d(phi_N,e):
    gcd,s,t=eea(phi_N,e)
    while t<0:
        t=t+phi_N
    return t
```

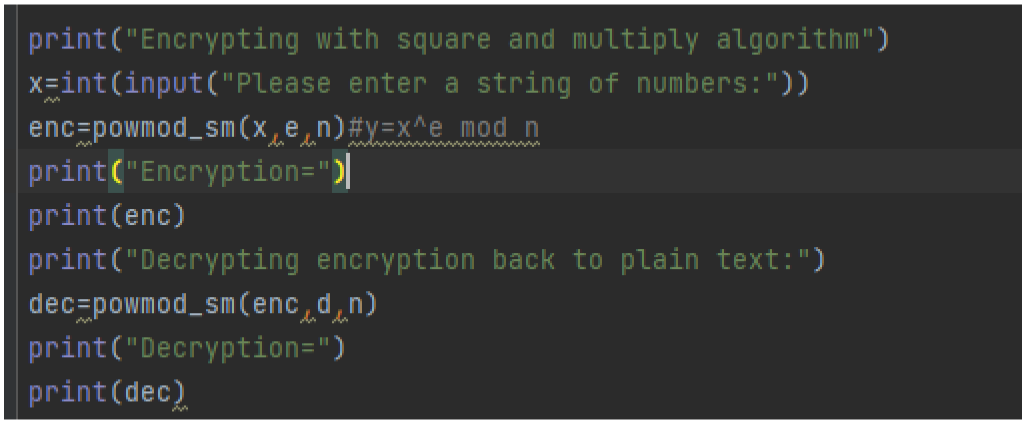


```
def eea(a, b):
    if a == 0 :
        return b,0,1
    gcd,si,ti = eea(b%a, a)
    s= ti - (b//a) * si
    t = si
    return gcd,s,t
```

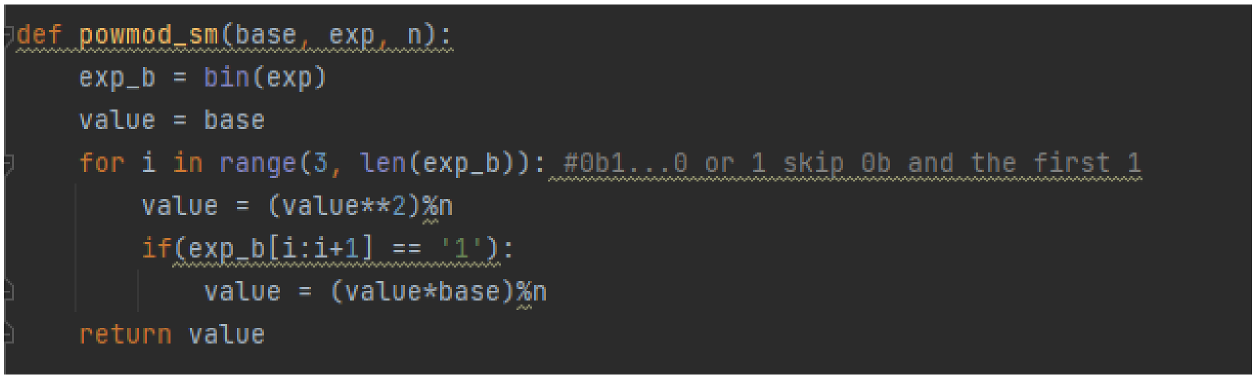


```
d=
50071222530003278302578342419047545958146820115478984675865491655182718646861682648110186977296991471925
```

5.Now the rest is for public and private key along with the code of powmod_sm

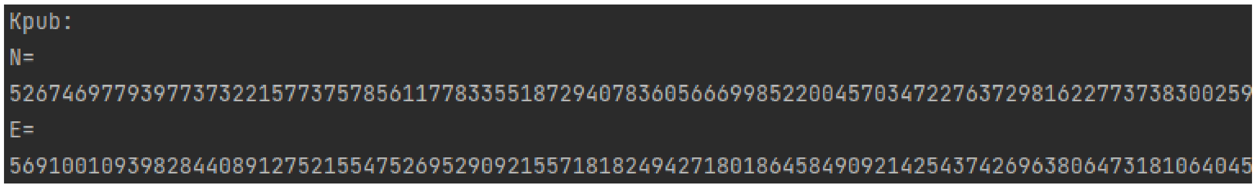


```
print("Encrypting with square and multiply algorithm")
x=int(input("Please enter a string of numbers:"))
enc=powmod_sm(x,e,n)#y=x^e mod n
print("Encryption=")
print(enc)
print("Decrypting encryption back to plain text:")
dec=powmod_sm(enc,d,n)
print("Decryption=")
print(dec)
```



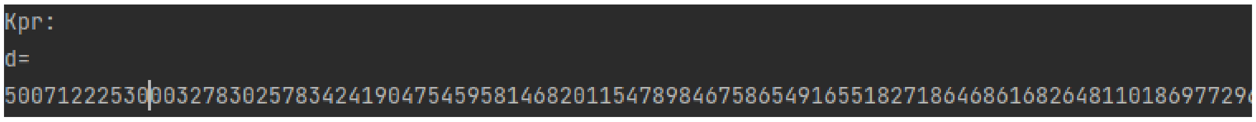
```
def powmod_sm(base, exp, n):
    exp_b = bin(exp)
    value = base
    for i in range(3, len(exp_b)): #0b1...0 or 1 skip 0b and the first 1
        value = (value**2)%n
        if(exp_b[i:i+1] == '1'):
            value = (value*base)%n
    return value
```

Out of public key



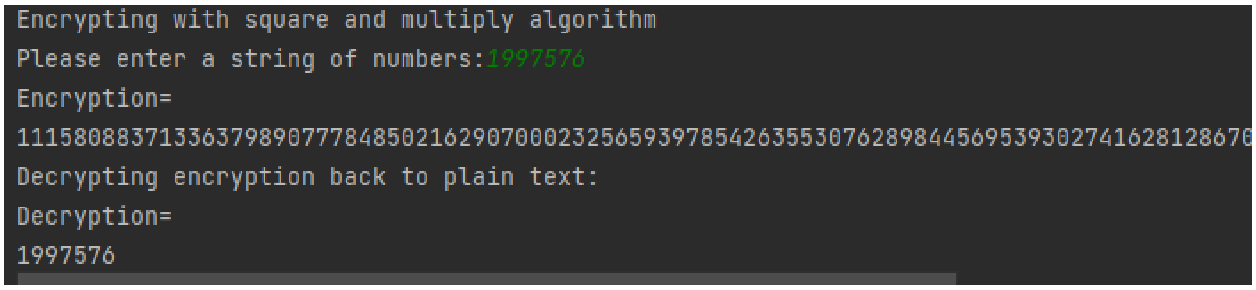
```
Kpub:
N=
5267469779397737322157737578561177833551872940783605666998522004570347227637298162277373830025
E=
5691001093982844089127521554752695290921557181824942718018645849092142543742696380647318106404
```

Out of private key



```
Kpr:
d=
5007122253000327830257834241904754595814682011547898467586549165518271864686168264811018697729
```

Taking in a user input of a string of numbers x and then encrypting it, and decrypting after to make sure it is working with powmod_sm or square and multiply algorithm



```
Encrypting with square and multiply algorithm
Please enter a string of numbers:1997576
Encryption=
1115808837133637989077784850216290700023256593978542635530762898445695393027416281286
Decrypting encryption back to plain text:
Decryption=
1997576
```

Note: for some reason sometimes I get an error when the program runs but most of the time it works. As I’ve only spent 2-3 days learning python I wasn’t well versed enough to debug it.