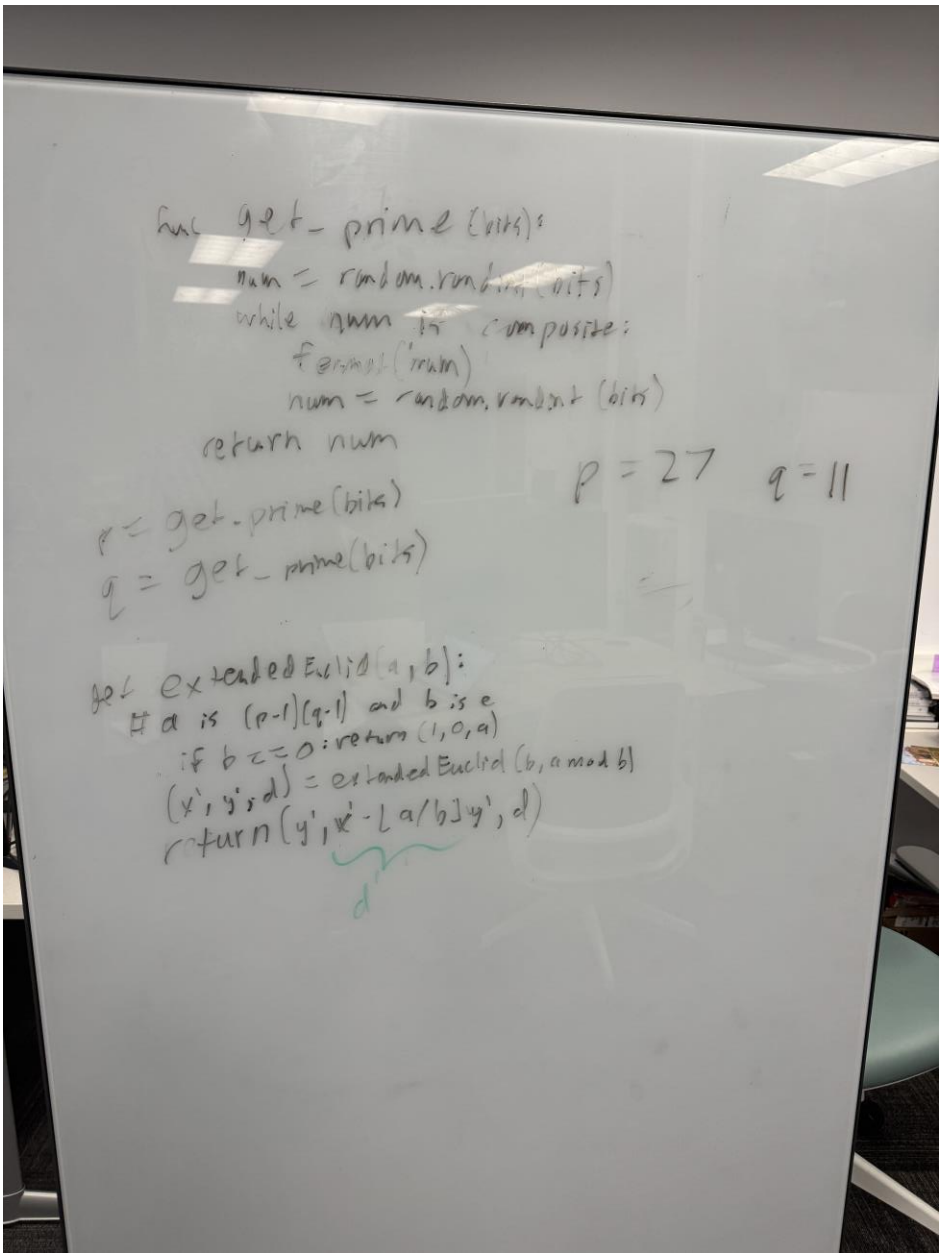
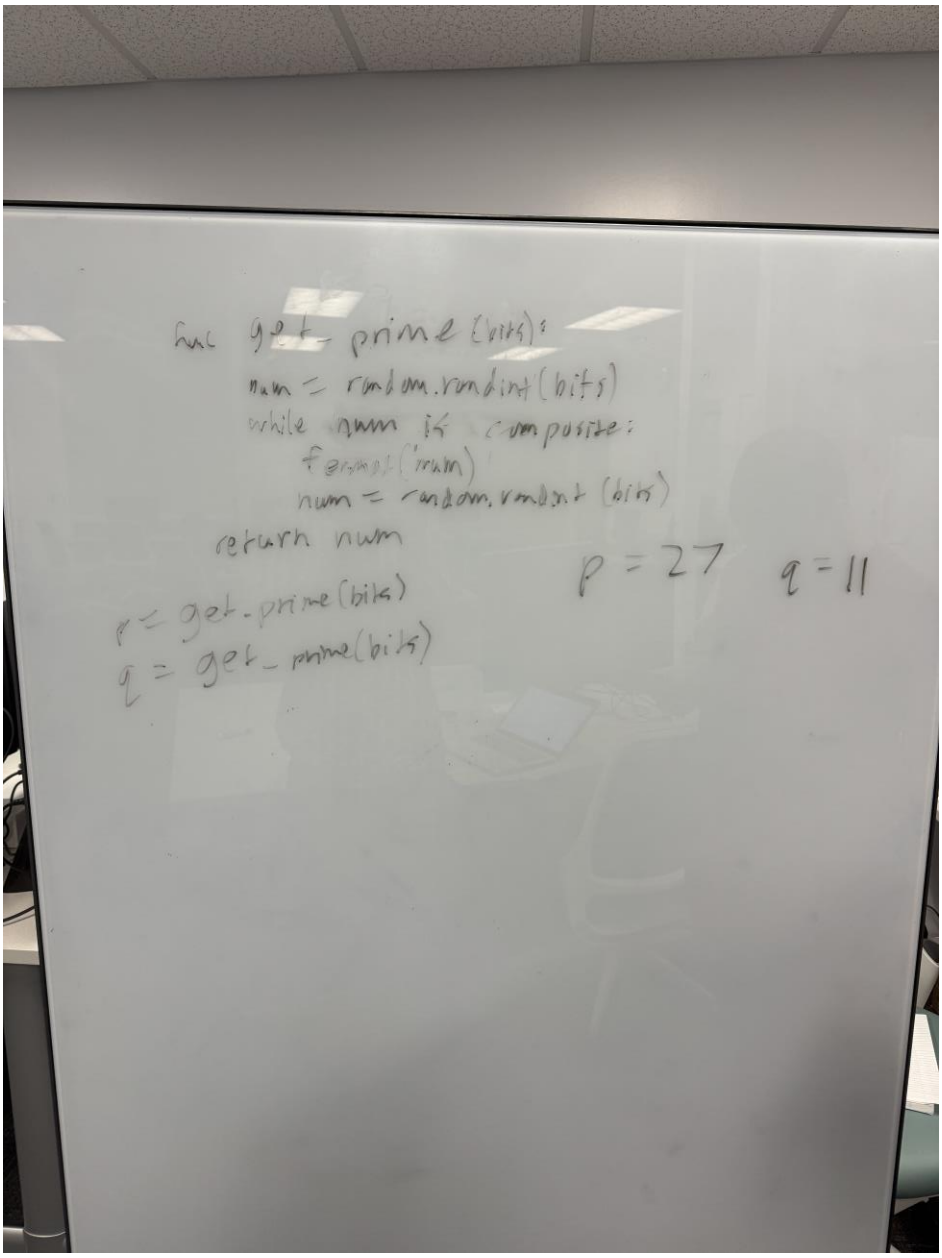
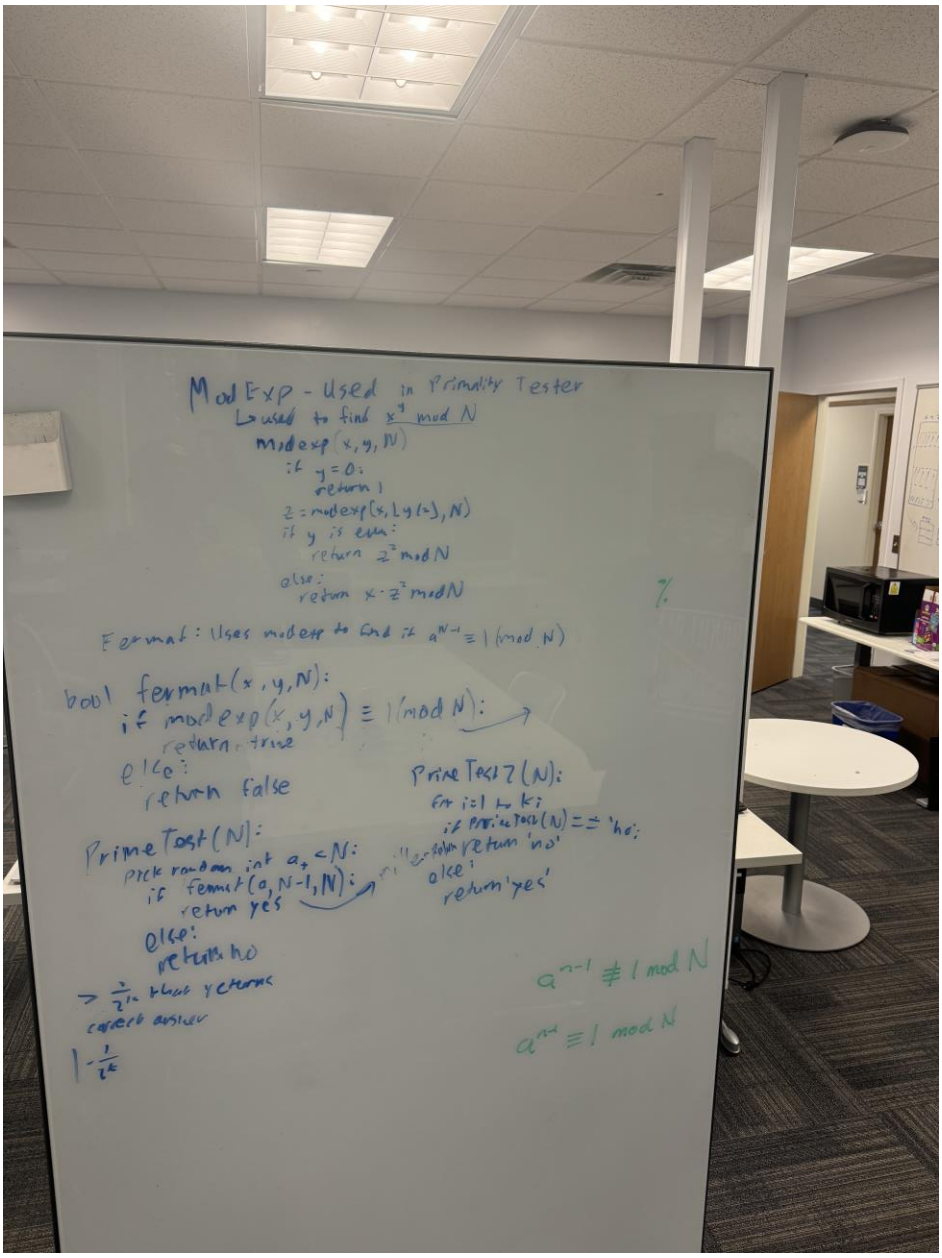
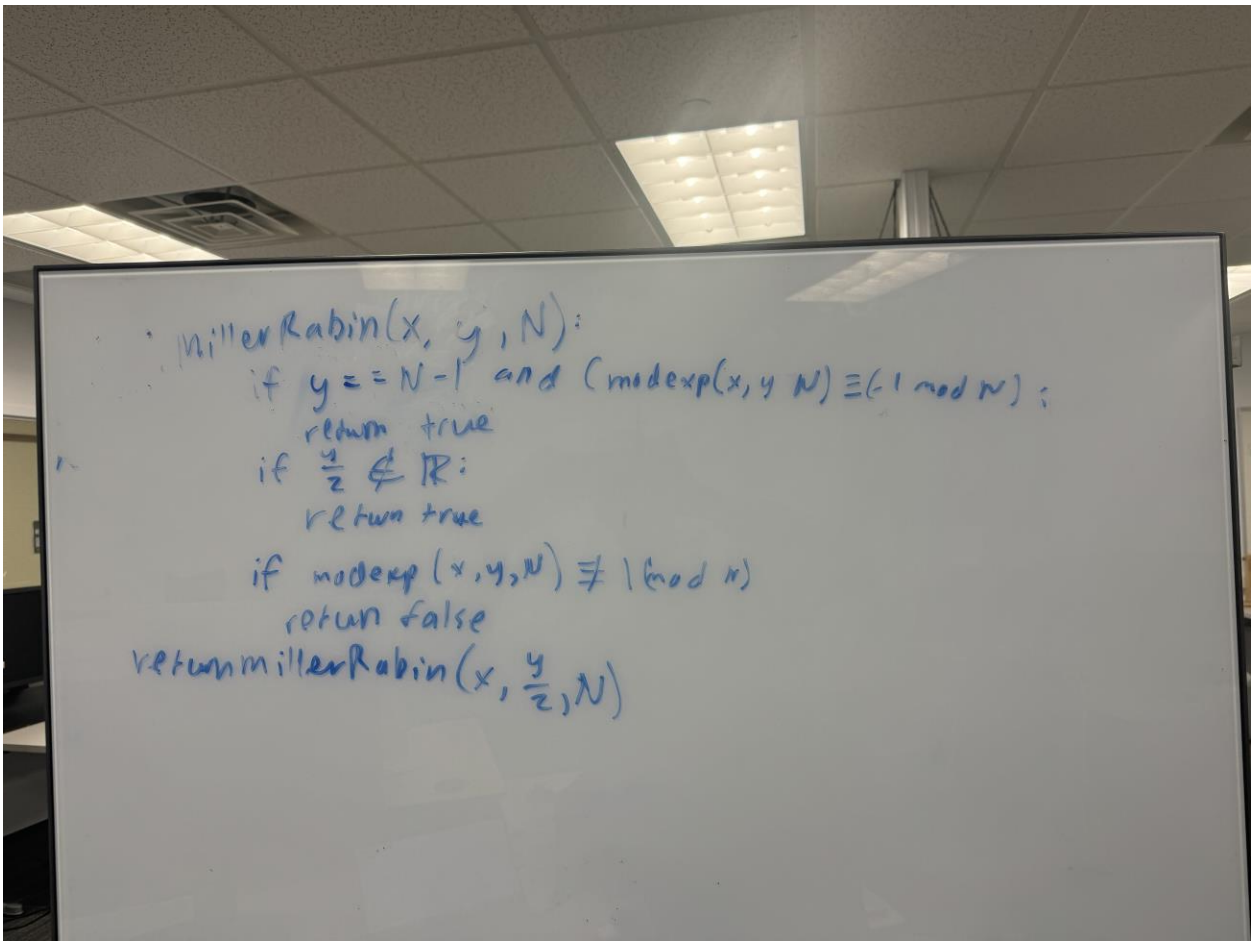

func get_prime(bits):
num = random.randint(bits)
while num is composite:
num = random.randint(bits)
return num
p = get_prime(bits)
q = get_prime(bits)
p = 27
q = 11
def extended Euclid(a, b):
# a is (p-1)(q-1) and b is e
if b == 0: return (1, 0, a)
(x', y', d) = extended Euclid(b, a mod b)
return (y', x' - ⌊a/b⌋y', d)
d


func get_prime(bits):
num = random.randint(bits)
while num is composite:
num = random.randint(bits)
return num
p = get_prime(bits)
q = get_prime(bits)
p = 27
q = 11


ModExp - Used in Primality Tester
↳ used to find $x^y \bmod N$
modexp(x, y, N)
if y = 0:
return 1
z = modexp(x, Ly/2J, N)
if y is even:
return $z^2 \bmod N$
else:
return $x \cdot z^2 \bmod N$
Fermat: Uses modexp to find if $a^{N-1} \equiv 1 \pmod N$
bool fermat(x, y, N):
if modexp(x, y, N) $\equiv 1 \pmod N$:
return true
else:
return false
PrimeTest(N):
pick random int $a < N$:
if fermat(a, N-1, N):
return yes
else:
return no
→ $\frac{1}{2}$ that returns correct answer
$1 - \frac{1}{2^k}$
PrimeTest2(N):
for i:1 to k:
if PrimeTest(N) == 'no':
return 'no'
else:
return 'yes'
$a^{n-1} \not\equiv 1 \bmod N$
$a^{n-1} \equiv 1 \bmod N$


millerRabin(x, y, N):
if y == N-1 and (modexp(x, y N) ≡ (-1 mod N):
return true
if y/2 ∉ ℕ:
return true
if modexp(x, y, N) ≢ 1 (mod N)
return false
return millerRabin(x, y/2, N)