Chinese remainder theorem

```
# A Python3 program to demonstrate
# working of Chinise remainder Theorem

# k is size of num[] and rem[].
# Returns the smallest number x
# such that:
# x % num[0] = rem[0],
# x % num[1] = rem[1],
# ..................
# x % num[k-2] = rem[k-1]
# Assumption: Numbers in num[]
# are pairwise coprime (gcd for
# every pair is 1)
def findMinX(num, rem, k):
	x = 1; # Initialize result

	# As per the Chinise remainder
	# theorem, this loop will
	# always break.
	while(True):

		# Check if remainder of
		# x % num[j] is rem[j]
		# or not (for all j from
		# 0 to k-1)
		j = 0;
		while(j < k):
			if (x % num[j] != rem[j]):
				break;
			j += 1;

		# If all remainders
		# matched, we found x
		if (j == k):
			return x;

		# Else try next number
		x += 1;

# Driver Code
num = [3, 4, 5];
rem = [2, 3, 1];
k = len(num);
print("x is", findMinX(num, rem, k));

# This code is contributed by mits
```

Extended euclidean

basic

```
# Python3 program to demonstrate Basic Euclidean Algorithm


# Function to return gcd of a and b
def gcd(a, b):
	if a == 0:
		return b

	return gcd(b % a, a)

# Driver code
if __name__ == "__main__":
a = 10
b = 15
print("gcd(", a, ",", b, ") = ", gcd(a, b))

a = 35
b = 10
print("gcd(", a, ",", b, ") = ", gcd(a, b))

a = 31
b = 2
print("gcd(", a, ",", b, ") = ", gcd(a, b))

# Code Contributed By Mohit Gupta_OMG <(0_o)>
```

Extended

```
# Python program to demonstrate working of extended
# Euclidean Algorithm

# function for extended Euclidean Algorithm


def gcdExtended(a, b):

	# Base Case
	if a == 0:
		return b, 0, 1

	gcd, x1, y1 = gcdExtended(b % a, a)

	# Update x and y using results of recursive
	# call
	x = y1 - (b//a) * x1
```

```
        y = x1

        return gcd, x, y


# Driver code
a, b = 35, 15
g, x, y = gcdExtended(a, b)
print("gcd(", a, ",", b, ") = ", g)
```

RSA

```
# Python for RSA asymmetric cryptographic algorithm.
# For demonstration, values are
# relatively small compared to practical application
import math


def gcd(a, h):
        temp = 0
        while(1):
                temp = a % h
                if (temp == 0):
                        return h
                a = h
                h = temp


p = 3
q = 7
n = p*q
e = 2
phi = (p-1)*(q-1)

while (e < phi):

        # e must be co-prime to phi and
        # smaller than phi.
        if(gcd(e, phi) == 1):
                break
        else:
                e = e+1

# Private key (d stands for decrypt)
# choosing d such that it satisfies
# d*e = 1 + k * totient
```

```
k = 2
d = (1 + (k*phi))/e

# Message to be encrypted
msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n
c = pow(msg, e)
c = math.fmod(c, n)
print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n
m = pow(c, d)
m = math.fmod(m, n)
print("Original Message Sent = ", m)


# This code is contributed by Pranay Arora.
```

Diffie Hellman

```
from random import randint

if __name__ == '__main__':

	# Both the persons will be agreed upon the
	# public keys G and P
	# A prime number P is taken
	P = 23

	# A primitive root for P, G is taken
	G = 9


	print('The Value of P is :%d'%(P))
	print('The Value of G is :%d'%(G))

	# Alice will choose the private key a
	a = 4
	print('The Private Key a for Alice is :%d'%(a))

	# gets the generated key
	x = int(pow(G,a,P))
```

```
# Bob will choose the private key b
b = 3
print('The Private Key b for Bob is :%d'%(b))

# gets the generated key
y = int(pow(G,b,P))


# Secret key for Alice
ka = int(pow(y,a,P))

# Secret key for Bob
kb = int(pow(x,b,P))

print('Secret key for the Alice is : %d'%(ka))
print('Secret Key for the Bob is : %d'%(kb))
```

Sha

```
# Python 3 code to demonstrate
# SHA hash algorithms.

import hashlib

# initializing string
str = "GeeksforGeeks"

# encoding GeeksforGeeks using encode()
# then sending to SHA256()
result = hashlib.sha256(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA256 is : ")
print(result.hexdigest())

print ("\r")

# initializing string
str = "GeeksforGeeks"

# encoding GeeksforGeeks using encode()
# then sending to SHA384()
result = hashlib.sha384(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA384 is : ")
print(result.hexdigest())
```

```
print ("\r")

# initializing string
str = "GeeksforGeeks"

# encoding GeeksforGeeks using encode()
# then sending to SHA224()
result = hashlib.sha224(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA224 is : ")
print(result.hexdigest())

print ("\r")

# initializing string
str = "GeeksforGeeks"

# encoding GeeksforGeeks using encode()
# then sending to SHA512()
result = hashlib.sha512(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA512 is : ")
print(result.hexdigest())

print ("\r")

# initializing string
str = "GeeksforGeeks"

# encoding GeeksforGeeks using encode()
# then sending to SHA1()
result = hashlib.sha1(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA1 is : ")
print(result.hexdigest())
```

Md5

```
# Python 3 code to demonstrate the
# working of MD5 (string - hexadecimal)

import hashlib
```

```
# initializing string
str2hash = "GeeksforGeeks"

# encoding GeeksforGeeks using encode()
# then sending to md5()
result = hashlib.md5(str2hash.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of hash is : ", end ="")
print(result.hexdigest())
```