

booths.py

```
from bitstring import BitArray
```

```
'''
```

Returns $m * r$ using Booth's algorithm.

$x = \text{len}(m)$ and $y = \text{len}(r)$. Note that this is the length in base 2.

```
'''
```

```
def booth(m, r, x, y):
```

```
    # Initialize
```

```
    totalLength = x + y + 1
```

```
    mA = BitArray(int = m, length = totalLength)
```

```
    rA = BitArray(int = r, length = totalLength)
```

```
    A = mA << (y+1)
```

```
    S = BitArray(int = -m, length = totalLength) << (y+1)
```

```
    P = BitArray(int = r, length = y)
```

```
    P.prepend(BitArray(int = 0, length = x))
```

```
    P = P << 1
```

```
    print "Initial values"
```

```
    print "A", A.bin
```

```
    print "S", S.bin
```

```
    print "P", P.bin
```

```
    print "Starting calculation"
```

```
    for i in range(1,y+1):
```

```
        if P[-2:] == '0b01':
```

```
            P = BitArray(int = P.int + A.int, length = totalLength)
```

```
            print "P + A:", P.bin
```

```
        elif P[-2:] == '0b10':
```

```
            P = BitArray(int = P.int + S.int, length = totalLength)
```

```
            print "P + S:", P.bin
```

```
        P = arith_shift_right(P, 1)
```

```

        print "P >> 1:", P.bin

    P = arith_shift_right(P, 1)

    print "P >> 1:", P.bin

    return P.int

def arith_shift_right(x, amt):
    l = x.len
    x = BitArray(int = (x.int >> amt), length = l)
    return x

if __name__ == "__main__":
    print("")

```

boothsClient.py

```

import socket,sys

print("")
print("Enter the multiplicand : ")
num1 = raw_input()

print("")
print("Enter the multiplier : ")
num2 = raw_input()

print("")
print("Enter the multiplicand's bit form's length : ")
num3 = raw_input()

print("")

```

```
print("Enter the multiplier's bit form's length : ")
```

```
num4 = raw_input()
```

```
numbers = num1 + "," + num2 + "," + num3 + "," + num4
```

```
clientsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
clientsocket.connect(("localhost",5000))
```

```
clientsocket.send(numbers)
```

```
print("")
```

```
clientsocket.close()
```

boothsServer.py

```
import
```

```
socket,sys,booth
```

```
s
```

```
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
serversocket.bind(("",5000))
```

```
serversocket.listen(5)
```

```
clientsocket, address = serversocket.accept()
```

```
receivedNumbersInKbs = clientsocket.recv(1024)
```

```
receivedNumbers = ""
```

```
while receivedNumbersInKbs :
```

```
    receivedNumbers = receivedNumbers + receivedNumbersInKbs
```

```
    receivedNumbersInKbs = ""
```

```
    receivedNumbersInKbs = clientsocket.recv(1024)
```

```
numberList = receivedNumbers.split(',')
```

```
multiplicand = int(numberList[0])  
multiplier = int(numberList[1])  
multiplicandLength = int(numberList[2])  
multiplierLength = int(numberList[3])
```

```
product =  
booths.booth(multiplicand,multiplier,multiplicandLength,multiplierLen  
gth)
```

```
print("")  
print("The product of " + str(multiplicand) + " and " +  
str(multiplier) + " is = " + str(product))  
print("")  
clientsocket.close()  
serversocket.close()
```

oddEvenIndex.html

```
<!DOCTYPE html>
<html lang="en">
<body>
    <h1>Enter elements (separated by ,) to be sorted : </h1>
    <form action="." method="POST">
        <input type="text" name="elements">
        <input type="submit" name="elementsForm" value="Enter">
    </form>

    <h2>{{ a }}</h2>
</body>
</html>
```

concurrentOddEven.py

```
from threading import Thread
```

```
def sort(x,l):
    for i in range(l, len(x)-1, 2):
        if x[i] > x[i+1]:
            x[i], x[i+1] = x[i+1], x[i]
            sorted = False
        if x[i] == x[i+1]:
            i+=2

def oddevensort(x):
    sorted = False
    while not sorted:
        sorted = True
        t = Thread(target = sort(x,0)) # for even
```

```

        t1=Thread(target = sort(x,1)) # for odd
        t.start()
        t1.start()
        t.join()
        t1.join()
    for i in range(0,len(x)-1):
        if x[i]<=x[i+1]:
            sorted = False
        else:
            x=oddevensort(x)
    return x

```

oddEven.py

```

from flask import Flask
from flask import request
from flask import render_template
import concurrentOddEven
app = Flask(__name__)

@app.route('/')
def my_form():
    return render_template("index.html")

@app.route('/', methods=['POST'])
def my_form_post():

    text = request.form["elements"]
    a = map(int, text.split(','))
    n = len(a)

```

```
concurrentOddEven.oddeven_sort(a)
```

```
return render_template("index.html", a=a)
```

```
if __name__ == '__main__':
```

```
    app.run()
```


plagiarismForm.html

```
<!DOCTYPE html>
<html lang="en">
<body>
    <h1>Enter the texts to be compared</h1>
    <form action="." method="POST">
        <input type="text" name="text1">
        <input type="text" name="text2">
        <input type="submit" name="my-form" value="Check !">
    </form>
</body>
</html>
```

plagiarismChecker.py

```
from flask import Flask
from flask import request
from flask import render_template
import stringComparison

app = Flask(__name__)

@app.route('/')
def my_form():
    return render_template("my-form.html")

@app.route('/', methods=['POST'])
def my_form_post():

    text1 = request.form['text1']
```

```

text2 = request.form['text2']

plagiarismPercent = stringComparison.extremelySimplePlagiarismChecker(text1,text2)

if plagiarismPercent > 50 :

    return "<h1>Plagiarism Detected !</h1>"

else :

    return "<h1>No Plagiarism Detected !</h1>"

```

```

if __name__ == '__main__':

    app.run()

```

stringComparison.py

```

def extremelySimplePlagiarismChecker(text1 , text2) :

    matches = 0

    smallerLength = 0

    if (len(text1) <= len(text2)) :

        smallerLength = len(text1)

    else :

        smallerLength = len(text2)

    i = 0

    while i < smallerLength :

        if text1[i] == text2[i] :

            matches = matches + 1

        i = i + 1

    similarityPercent = (matches/smallerLength) * 100

    return similarityPercent

```

```

if __name__ == "__main__":

    print("")

```

sha.py

```
def chunks(bits, chunkSize) :
```

```
    return [bits[i:i+chunkSize] for i in range(0, len(bits), chunkSize)] # slice & dice
```

```
def rotateLeft(bits, positions) :
```

```
    return ((bits << positions) | (bits >> (32 - positions))) & 0xffffffff # apparently this how it's done
```

```
def shaDigest(data) :
```

```
    # slices of the digest
```

```
    h0 = 0x67452301
```

```
    h1 = 0xEFCDAB89
```

```
    h2 = 0x98BADCFE # reverse of h1
```

```
    h3 = 0x10325476 # reverse of h0
```

```
    h4 = 0xC3D2E1F0 # end pairings
```

```
# so first things first we need to get the input characters -> int form -> bit form
```

```
    bitFormOfData = ""
```

```
    for c in range(len(data)) :
```

```
        bitFormOfData = bitFormOfData + '{0:08b}'.format(ord(data[c]))
```

```
        # ord does this -> ord('a') returns 97
```

```
        # the {0:08b} formats it into 8 bits so 9 is 00001001 instead of 1001
```

```
    originalBitFormOfData = bitFormOfData # needed later for appending
```

```
    bitFormOfData = bitFormOfData + "1" # for some strange reason, 1 is appended
```

```
# now we need to pad 0's till len(bitFormOfData) = 448 mod 512
```

```
while ((len(bitFormOfData) % 512) != 448) :
```

```
    bitFormOfData = bitFormOfData + "0"
```

```
# appending original message so length of it would be an exact multiple of 512
# also format it to 64-bit representation
bitFormOfData = bitFormOfData + '{0:064b}'.format(len(originalBitFormOfData))
```

```
# essentially, this is what's being done ->
```

```
# break bitFormOfData into 512 slices
```

```
# break 32 bit slices from a single 512 bit slice - this is our word
```

```
# effectively, we are TRANSFORMING each 32-bit word into a 80-bit list (i dunno why)
```

```
# now operations will be done on this 80-bit list called w
```

```
for c in chunks(bitFormOfData , 512) : # c is a slice of 512 bits from bitFormOfData
```

```
    words = chunks(c,32) # words is a slice of 32 bits from c => there will be 16 such slices
```

```
    w = [0] * 80 # w = [0,0,0,0,0,.....(80 times)] -> it's a list
```

```
    for n in range(0,16) :
```

```
        w[n] = int(words[n] , 2) # prototype of int(x,base = 10) , here base = 2 for binary
```

```
    for i in range(16,80) :
```

```
        w[i] = rotateLeft((w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16]), 1) # ^ is for XOR
```

```
a = h0
```

```
b = h1
```

```
c = h2
```

```
d = h3
```

```
e = h4
```

```
# the k values could be anything (but the receiver should also have the same ones in a digital
signature scenario)
```

```
for i in range(0,80) :
```

```
    if (0 <= i <= 19) :
```

```
        f = b ^ c ^ d # f here is a function
```

```

    k = 0x5A827999 # could be anything
elif (20 <= i <= 39) :
    f = b ^ c ^ d # the idea is that for each range there will be a different f
    k = 0x6ED9EBA1 # could be anything
elif (40 <= i <= 59) :
    f = b ^ c ^ d # f could be b ^ c & d OR !c ^ d - or whatever
    k = 0x8F1BBCDC # could be anything
elif (60 <= i <= 79) :
    f = b ^ c ^ d # lazy much,hence same function everywhere
    k = 0xCA62C1D6 # could be anything

```

```

temp = rotateLeft(a, 5) + f + e + k + w[i] & 0xffffffff
e = d
d = c
c = rotateLeft(b , 30)
b = a
a = temp

```

```

h0 = h0 + a
h1 = h1 + b
h2 = h2 + c
h3 = h3 + d
h4 = h4 + e

```

```

return '%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4)

```

```

if __name__ == "__main__": # so that it can be imported
    print("Enter a message : ")
    message = raw_input()
    print("")
    digest = shaDigest(message)

```

```
print("It's SHA-1 digest is -> ")
print(digest)
print("")
```

shaClient.py

```
import socket,sys,sha
```

```
print("")
print("Enter your password to gain a Kerberos ticket : ")
password = raw_input()
```

```
clientsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
clientsocket.connect(("localhost",5000))
```

```
passwordDigest = sha.shaDigest(password)
clientsocket.send(passwordDigest)
```

```
print("")
clientsocket.close()
```

shaServer.py

```
import socket,sys,sha
```

```
print("")
print("Enter password registered with Kerberos Server : ")
passwordInKerberosServer = raw_input()
```

```
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.bind(("",5000))
```

```
serversocket.listen(5)
```

```
clientsocket, address = serversocket.accept()
```

```
receivedPasswordDigestInKbs = clientsocket.recv(1024)
```

```
receivedPasswordDigest = ""
```

```
while receivedPasswordDigestInKbs :
```

```
    receivedPasswordDigest = receivedPasswordDigest + receivedPasswordDigestInKbs
```

```
    receivedPasswordDigestInKbs = ""
```

```
    receivedPasswordDigestInKbs = clientsocket.recv(1024)
```

```
registeredPassword = sha.shaDigest(passwordInKerberosServer)
```

```
print("")
```

```
if registeredPassword == receivedPasswordDigest :
```

```
    print("Kerberos ticket granted !")
```

```
else :
```

```
    print("Kerberos ticket denied !")
```

```
print("")
```

```
clientsocket.close()
```

```
serversocket.close()
```


8queens.json

```
{"matrix": [ [1,0,0,0,0,0,0,0],
             [1,0,0,0,0,0,0,0],
             [1,0,0,0,0,0,0,0],
             [1,0,0,0,0,0,0,0],
             [1,0,0,0,0,0,0,1],
             [1,0,0,0,0,0,0,0],
             [1,0,0,0,0,0,0,0],
             [1,0,0,0,0,0,0,0]] }
```

8queens.py

```
import json
N=8
queens = [-1,-1,-1,-1,-1,-1,-1,-1]
def isValid(col,row):
    for i in range(col):
        if(queens[i] == row):
            return False
    for i in range(col):
        if(abs(col-i) == abs(row-queens[i])):
            return False
    return True

def place(n):
    placed = False;
    for i in range(N):
        if(isValid(n,i)):
            queens[n] = i
            placed=True
```

```
        break
```

```
    if(not placed):
```

```
        return False
```

```
    if( n == (N-1)):
```

```
        return True
```

```
    while(not place(n+1)):
```

```
        placed = False
```

```
        for i in range(queens[n]+1,N):
```

```
            if(isValid(n,i)):
```

```
                queens[n] = i
```

```
                placed=True;
```

```
                break
```

```
        if(not placed):
```

```
            return False
```

```
    return True
```

```
inputFile = open("8queens.json")
```

```
data = json.loads(inputFile.read())
```

```
data=data["matrix"]
```

```
for i in data:
```

```
    print(i)
```

```
print "\n"
```

```
for i in range(0,N):
```

```
    if(data[i][0] == 1):
```

```
        queens[0] = i
```

```
place(1)
```

```
print queens

print "\n"

outfile = open("out.json","w")

data = {}

matrix = [[[],[],[],[],[],[],[],[]]]

for i in range(0,N) :
    for j in range(0,N):
        if(queens[j] == i):
            matrix[i].insert(j,1)
        else:
            matrix[i].insert(j,0)

data["matrix"] = matrix

json.dump(data,outfile)

for i in matrix:
    print(i)
```


binarySearch.py

```
def bubbleSort(a,n) :
```

```
    i = 0
```

```
    while i < n :
```

```
        j = i + 1
```

```
        while j < n :
```

```
            if a[i] > a[j] :
```

```
                temp = a[i]
```

```
                a[i] = a[j]
```

```
                a[j] = temp
```

```
            j = j + 1
```

```
        i = i + 1
```

```
print("Enter size of array : ")
```

```
n = int(input())
```

```
print("Enter unsorted elements of array : ")
```

```
a = [] # take care - python doesn't have arrays (lists are used instead)
```

```
i = 0
```

```
while i<n :
```

```
    a.append(int(input()))
```

```
    i = i + 1
```

```
# a.sort() # sort is built-in for lists - a better option is sorted
```

```
# since size matters, a bubbleSort function is added but a.sort() is more optimal
```

```
bubbleSort(a,n)
```

```
print("Enter number to be searched : ")
```

```
key = int(input())
```

```
left = 0
```

```
right = n - 1
```

```
mid = (left + right) // 2 # Floor Division
```

```
if key not in a:
```

```
    print("Key is absent !")
```

```
else:
```

```
    while left <= right :
```

```
        if a[mid] == key :
```

```
            print("Key was found at position = " + str(mid+1))
```

```
            break
```

```
        elif a[mid] < key :
```

```
            left = mid + 1
```

```
        else :
```

```
            right = mid - 1
```

```
mid = (left + right) // 2 # Floor Division
```

createXMLfile.py

```
import os
import random
import sys
name = raw_input("Enter the name of file: ").xml"
file = open(name,'w')
file.write("<Numbers>\n")
for i in range(10):
    file.write("\t<integer num = \"\""+str(random.randint(0,800))+ "\" ></integer>\n")
file.write("</Numbers>\n")
file.close()
temp =input(name+" file is written & press 1 to see its contents:")
if(temp==1):
    os.system("cat "+name)
    tmp=input("\npress 2 to start quickstart program: ")
    if(tmp==2):
        os.system("python quickSort.py "+name)
```

quicksort.py

```
from xml.etree import ElementTree
import random
import sys
from threading import Thread, current_thread
#=====
def read_xml(file_path):
    doc = ElementTree.parse(file_path)
    root = doc.getroot()
    arr = []
```

```

for value in root.iter('integer'):
    arr.append(int(value.attrib.values()[0]))

return arr

#=====

def swap(arr,left,right):
    temp = arr[left]
    arr[left] = arr[right]
    arr[right] = temp

#=====

def partition(arr,low,high):
    try:
        left=low
        pivot = arr[low]
        right = high
        done = False
        while not done:
            while left <= right and arr[left] <= pivot:
                left = left + 1
            while arr[right] >= pivot and right >=left:
                right = right -1
            if right < left:
                done= True
            else:
                swap(arr,left,right)
                swap(arr,low,right)
    except IndexError:
        print ""

    return right

#=====

def quicksort(arr,low,high):
    if low<high:

```



```
        pivot = partition(arr, low, high)

        t=Thread(target = quicksort, args = (arr, low, pivot - 1))

        t.start()

        t.join()

        t1=Thread(target = quicksort, args = (arr, pivot + 1, high))

        t1.start()

        t1.join()

    return arr

#=====

arr = read_xml(sys.argv[1]);

arr1 = quicksort(arr, 0, len(arr)-1)

print arr1

#=====
```


diningPhilosophers.py

```
from pymongo import MongoClient
```

```
client = MongoClient()
```

```
import sys,threading,time
```

```
db = client.diningPhilosophers # the name of the database
```

```
collection = db.diningPhilosophersCollection # name of the collection
```

```
doc0 = {
```

```
    "number" : 0,
```

```
    "name" : "Descartes",
```

```
    "thought" : "A donut's hope proves it's existence."
```

```
}
```

```
doc1 = {
```

```
    "number" : 1,
```

```
    "name" : "Marx",
```

```
    "thought" : "Everybody desires donuts."
```

```
}
```

```
doc2 = {
```

```
    "number" : 2,
```

```
    "name" : "Aristotle",
```

```
    "thought" : "A donut contains it's donut-ness."
```

```
}
```

```
doc3 = {
```

```
    "number" : 3,
```

```
    "name" : "Hume",
```

```
    "thought" : "Donuts exist because I imagine donuts."
```

```
}
```

```

doc4 = {
    "number" : 4,
    "name" : "Nietzsche",
    "thought" : "Stop at nothing to get your donut."
}

doc_id = collection.insert_one(doc0) # the variable doc_id is not needed
doc_id = collection.insert_one(doc1)
doc_id = collection.insert_one(doc2)
doc_id = collection.insert_one(doc3)
doc_id = collection.insert_one(doc4)
#doc1_id = collection.insert_one(doc1)

#doc2 = collection.find_one({"number" : 1})
#print(doc2)

#myValues[i] = bla['value'] # getting specific value from a document

```

```

class Semaphore(object):

    def __init__(self, initial):
        self.lock = threading.Condition(threading.Lock())
        self.value = initial

    def up(self):
        with self.lock:
            self.value += 1
            self.lock.notify()

    def down(self):
        with self.lock:

```

```
while self.value == 0:
    self.lock.wait()
    self.value -= 1
```

```
class ChopStick(object):
```

```
    def __init__(self, number):
```

```
        self.number = number      # chop stick ID
        self.user = -1            # keep track of philosopher using it
        self.lock = threading.Condition(threading.Lock())
        self.taken = False
```

```
    def take(self, user):        # used for synchronization
```

```
        with self.lock:
            while self.taken == True:
                self.lock.wait()
            self.user = user
            self.taken = True
            sys.stdout.write("p[%s] took c[%s]\n" % (user, self.number))
            self.lock.notifyAll()
```

```
    def drop(self, user):        # used for synchronization
```

```
        with self.lock:
            while self.taken == False:
                self.lock.wait()
            self.user = -1
            self.taken = False
            doc = collection.find_one({"number" : user})
            sys.stdout.write("p[%s] i.e. %s dropped c[%s] and thinks -> %s\n" % (user, doc["name"],
self.number, doc["thought"]))
            self.lock.notifyAll()
```

```

class Philosopher (threading.Thread):

    def __init__(self, number, left, right, butler):
        threading.Thread.__init__(self)
        self.number = number      # philosopher number
        self.left = left
        self.right = right
        self.butler = butler

    def run(self):
        for i in range(20):
            self.butler.down()      # start service by butler
            time.sleep(0.1)         # think
            self.left.take(self.number) # pickup left chopstick
            time.sleep(0.1)         # (yield makes deadlock more likely)
            self.right.take(self.number) # pickup right chopstick
            time.sleep(0.1)         # eat
            self.right.drop(self.number) # drop right chopstick
            self.left.drop(self.number) # drop left chopstick
            self.butler.up()        # end service by butler
            sys.stdout.write("p[%s] finished thinking and eating\n" % self.number)

def main():
    # number of philosophers / chop sticks
    n = 5

    # butler for deadlock avoidance (n-1 available)
    butler = Semaphore(n-1)

```

```
# list of chopsticks
```

```
c = [ChopStick(i) for i in range(n)]
```

```
# list of philosophers
```

```
p = [Philosopher(i, c[i], c[(i+1)%n], butler) for i in range(n)]
```

```
for i in range(n):
```

```
    p[i].start()
```

```
if __name__ == "__main__":
```

```
    main()
```


dsaMillerRabin.py

```
import random
```

```
def numberOfBits(p):  
    return (len(bin(p))-2)
```

```
def brutalPrime(n) :  
    isPrime = True  
    i = 2  
    while (i < n/2) : # RS Agarwal stuff - surprisingly i recalled it  
        if n % i == 0 :  
            isPrime = False  
            return isPrime  
        i = i + 1  
  
    return isPrime
```

```
def pseudoPrimeMillerRabin(n) :  
    k = 1000 # miller-rabin accuracy  
    i = 0  
    isProbablyPrime = True  
    while i < k :  
        randomCheck = random.randint(2,(n-2))  
        if (n % randomCheck) == 0 :  
            isProbablyPrime = False  
            return isProbablyPrime  
        i = i + 1  
  
    return isProbablyPrime
```

```

a = []
print("")
print("Enter the parameter tuple (p,q,g) : ")
print("Enter value for p : ")
p = long(raw_input())
print("Enter value for q : ")
q = long(raw_input())
print("Enter value for g : ")
g = long(raw_input())
print("")
print("Here's how good the DSA tuple is -> ")
print("According to brute-force primality testing is q prime ?")
print(brutalPrime(q))
print("")
print("According to Miller-Rabin primality testing, q is most probably prime ?")
print(pseudoPrimeMillerRabin(q))
print("")
numberOfBitsOfQ = numberOfBits(q)
print("Is number of bits of q = 160 ?")
if numberOfBitsOfQ == 160 :
    print("True")
else :
    print("False")

print("")
print("Does q divide (p-1) ?")
if ((p-1) % q) == 0 :
    print("True")
else :
    print("False")

```

```

print("")
print("Is number of bits of p between 512 & 1024 ? ")
numberOfBitsOfP = numberOfBits(p)
if ((numberOfBitsOfP >= 512) and (numberOfBitsOfP <= 1024)) :
    print("True")
else :
    print("False")

print("")
print("Is g of the right form i.e.  $(h^{((p-1)/q)} \bmod p)$  where  $h = 2$  ?")
h = 2
complexExponent = ((p-1) / q)
if (pow(h,complexExponent) % p) == g :
    print("True")
else :
    print("False")
print("")

```


passwordEncryption.py

```
import base64
```

```
def vigenereEncryptionUsedToMeanSomethingBackInTheDay(key,message) :  
    cipherTextCharacters = []  
    for i in range(len(message)) :  
        keyForCurrentCharacter = key[(i % len(key))] # the % 'wraps' the key over the message  
        repeatedly  
  
        currentCipherTextCharacter = chr(ord(message[i]) + ord(keyForCurrentCharacter) % 256)  
        # this is what the built-in functions chr() and ord() do ->  
        # chr(97) returns 'a'  
        # ord('a') returns 97  
        # keep in mind that 'a' is a string of length = 1 and not chars  
  
        cipherTextCharacters.append(currentCipherTextCharacter)  
  
    cipherText = "".join(cipherTextCharacters) # combines the characters in the character array to  
    form a string  
    return cipherText  
  
    # urlsafe_b64encode - standard encoding of the string so that it can be sent SAFELY as part of a  
    URL/mail etc
```

```
def vigenereDecryptionUsedToMeanSomethingBackInTheDay(key,cipherText) :  
    decryptedTextCharacters = []  
    for i in range(len(cipherText)) :  
        keyForCurrentCharacter = key[(i % len(key))]  
  
        currentDecryptedTextCharacter = chr((256 + ord(cipherText[i]) - ord(keyForCurrentCharacter))  
        % 256) # observe the order of operations  
  
        decryptedTextCharacters.append(currentDecryptedTextCharacter)  
  
    decryptedText = "".join(decryptedTextCharacters)  
  
    return decryptedText # original message so no need of urlsafe_b64encode
```

```
print("Enter your password (i.e. key) - ")
key = raw_input()
print("")
print("Enter your message to be encrypted - ")
message = raw_input()
print("")
print("This is the cipher text generated - which can totally not be cracked ;) - ")
cipherText = vigenereEncryptionUsedToMeanSomethingBackInTheDay(key,message)
gibberish = base64.urlsafe_b64encode(cipherText) # if cipherText is printed directly, you cannot see
the actual characters
print(gibberish)
print("")
print("This is the decrypted message using your password which is supposed to match your original
message - ")
decryptedText = vigenereDecryptionUsedToMeanSomethingBackInTheDay(key,cipherText)
print(decryptedText)
print("")
```

prng.py

```
seed = 123456789 # any random number will do
```

```
a = 1103515245
```

```
c = 12345
```

```
m = pow(2,32)
```

```
# Linear Congruential Generator is the simplest & oldest
```

```
# for LCG, the values of a,c & m may be anything but for it's full potential (i.e. longest unrepeated sequence) ->
```

```
# c & m should be co-prime (means they have no common factor other than 1)
```

```
# (a-1) should be divisible by all prime factors of m
```

```
def LCG() :
```

```
    global seed # Python would otherwise create a local seed in the next statement instead of using the global seed
```

```
    seed = (a * seed + c) % m
```

```
    return seed
```

```
print("How many random numbers do you want ? ")
```

```
n = int(raw_input())
```

```
print("")
```

```
i = 0
```

```
while i<n :
```

```
    print(LCG())
```

```
    i = i + 1
```

```
print("")
```


trigonometryActivity.java

```
package com.example.khalid.trigocalc;
```

```
import android.graphics.Color;
```

```
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
import android.widget.Button;
```

```
import android.widget.TextView;
```

```
import android.view.View; // need to add this separately
```

```
public class CalcActivity extends AppCompatActivity
```

```
{
```

```
    // global stuff here
```

```
    TextView result;
```

```
    String currentNumber = "";
```

```
    double answer = 0;
```

```
    void numberPressed(int n)
```

```
    {
```

```
        currentNumber += String.valueOf(n);
```

```
        result.setText(currentNumber);
```

```
    }
```

```
    public enum Operation
```

```
    {
```

```
        SIN,COS,TAN,EQUALS
```

```
    }
```

```

void processOperation(Operation operation)
{
    if(currentNumber != null)
    {
        switch (operation)
        {
            case SIN :
                answer = Math.sin(Math.toRadians(Integer.parseInt(currentNumber)));
                break;

            case COS :
                answer = Math.cos(Math.toRadians(Integer.parseInt(currentNumber)));
                break;

            case TAN :
                answer = Math.tan(Math.toRadians(Integer.parseInt(currentNumber)));
                break;

            case EQUALS :
                result.setText(String.valueOf(answer));
                break;
        }
    }
}

```

@Override

```

protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_calc);
}

```

```
Button zero = (Button)findViewById(R.id.zero);
Button one = (Button)findViewById(R.id.one);
Button two = (Button)findViewById(R.id.two);
Button three = (Button)findViewById(R.id.three);
Button four = (Button)findViewById(R.id.four);
Button five = (Button)findViewById(R.id.five);
Button six = (Button)findViewById(R.id.six);
Button seven = (Button)findViewById(R.id.seven);
Button eight = (Button)findViewById(R.id.eight);
Button nine = (Button)findViewById(R.id.nine);
Button equals = (Button)findViewById(R.id.equals);
Button clear = (Button)findViewById(R.id.clear);
final Button sin = (Button)findViewById(R.id.sin);
final Button cos = (Button)findViewById(R.id.cos);
final Button tan = (Button)findViewById(R.id.tan);
result = (TextView)findViewById(R.id.result);
```

```
result.setText("");
```

```
zero.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        numberPressed(0);
    }
});
```

```
one.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        numberPressed(1);
    }
});
```

```
});
```

```
two.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        numberPressed(2);  
    }  
});
```

```
three.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        numberPressed(3);  
    }  
});
```

```
four.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        numberPressed(4);  
    }  
});
```

```
five.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        numberPressed(5);  
    }  
});
```

```
six.setOnClickListener(new View.OnClickListener() {
```

```
@Override  
public void onClick(View v) {  
    numberPressed(6);  
}  
});
```

```
seven.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        numberPressed(7);  
    }  
});
```

```
eight.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        numberPressed(8);  
    }  
});
```

```
nine.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        numberPressed(9);  
    }  
});
```

```
equals.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        sin.setBackgroundColor(Color.parseColor("#ffffff"));  
    }  
});
```

```
        cos.setBackgroundColor(Color.parseColor("#ffffff"));
        tan.setBackgroundColor(Color.parseColor("#ffffff"));
        processOperation(Operation.EQUALS);
    }
});
```

```
sin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        sin.setBackgroundColor(Color.parseColor("#ff33b5e5"));
        processOperation(Operation.SIN);
    }
});
```

```
cos.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        cos.setBackgroundColor(Color.parseColor("#ff33b5e5"));
        processOperation(Operation.COS);
    }
});
```

```
tan.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        tan.setBackgroundColor(Color.parseColor("#ff33b5e5"));
        processOperation(Operation.TAN);
    }
});
```

```
clear.setOnClickListener(new View.OnClickListener() {
```

```
@Override  
public void onClick(View v) {  
    answer = 0;  
    currentNumber = "";  
    result.setText("0");  
}  
});  
  
}  
}
```