import numpy as np

import matplotlib as mpl

import cmath

from itertools import permutations

def initialize\_basis():

basis\_0=np.array([[1],[0]])

basis\_1=np.array([[0],[1]])

return basis\_0,basis\_1

def initialize\_qubit()

alpha\_real=int(input("enter real no of alpha"))

alpha\_img=int(input("enter imaginary part of alpha"))

beta\_real=int(input("enter real part of beta"))

beta\_img=int(input("enter imaginary part of beta"))

alpha=complex(alpha\_real,alpha\_img)

beta=complex(beta\_real,beta\_img)

def to\_ket(alpha,beta):

ket=np.array([[alpha],[beta]])

return ket

def to\_bra():

ket=to\_ket()

bra=ket.conjugate().transpose()

return bra

def inner\_product(m1,m2):

inner\_pro=np.dot(m1,m2)

return inner\_pro

def check\_validity():

inner\_pro=inner\_product(alpha,beta)

if inner\_pro==1:

print('valid')

return 1

else:

print('invalid')

def construct\_standard\_basis(n):

perm=permutations([1,0],n)

lst=list(perm)

for i in lst

if (sum(lst[i]!=1)):

lst.remove(lst[i])

return lst

def combine\_qbits():

tensor\_product=np.tensordot((alpha,beta) axes=([1,0],[0,1]))

return tensor\_product

def measure\_single():

prob\_1=alpha\*\*2

prob\_2=beta\*\*2

def measure\_multi(qubit\_state):

no=random.randint(0,n)

lst1=lst[no]

measure\_mqubit=np.dot(lst1,qubit\_state)

return measure\_mqubit