In [138]:

```
# required Imports
import math
import cmath
import numpy as np
from numpy.random import choice

def get_ground_state(num_qubits):
    # return vector of size 2**num_qubits with all zeroes except first element which is 1
    # validating Inputs by user
    assert type(num_qubits) == int , 'Input Must be a integer'
    assert num_qubits > 0 , 'Input must be greater than 0'


    q1 = np.zeros((2**num_qubits))
    q1[0]=1
    return q1

def get_operator(total_qubits, gate_unitary, target_qubits,params):
    # return unitary operator of size 2**n x 2**n for given gate and target qubits
    # validating Inputs by user
    assert gate_unitary=='u3'or gate_unitary=='x'or gate_unitary=='h'or gate_unitary=='c
x', 'Input Gate Undefiend'
    if gate_unitary=='cx':
        assert len(target_qubits)==2,'2 Target Qubits required'
    else:
        assert len(target_qubits)==1,'1 Target Qubits required'
    if gate_unitary=='u3':
        assert len(params)==3,'3 parameters required'
    for i in range(len(target_qubits)):
        assert target_qubits[i]>=0 and target_qubits[i]<total_qubits , 'Target Qubits Ou
t of Bound'



    #Cached Reqularly used gates
    li = {
        'h': np.array([
        [1/np.sqrt(2), 1/np.sqrt(2)],
        [1/np.sqrt(2), -1/np.sqrt(2)]
        ]),
        'x': np.array([
        [0, 1],
        [1, 0]
        ])
        }
    if(gate_unitary=='u3'):
        theta = params['theta']
        phi = params['phi']
        lamb = params['lamb']
        i = complex(0+1j)
        a1 = math.cos(theta/2)
        a2 = -1*cmath.exp(i*lamb)*math.sin(theta/2)
        a3 = cmath.exp(i*phi)*math.sin(theta/2)
        a4 = cmath.exp(i*lamb+i*phi)*math.sin(theta/2)

        li['u3'] = np.array([
        [a1, a2],
        [a3, a4]
        ])


    I = np.identity(2)

    #calcualte the operator
    if(len(target_qubits)==1):
        gate = li[gate_unitary]
        if(total_qubits==1):
```

```python
            return gate


        if(target_qubits[0]==0):
            O = np.kron(gate,I)
        elif(target_qubits[0]==1):
            O = np.kron(I,gate)
        else:
            O = np.kron(I,I)
        for i in range(2,total_qubits):
            if(i==target_qubits[0]):
                O = np.kron(O,gate)
            else:
                O = np.kron(O,I)

        return O
    else:
        P0x0 = np.array([
        [1, 0],
        [0, 0]
        ])
        P1x1 = np.array([
        [0, 0],
        [0, 1]
        ])
        X = li['x']
        if(target_qubits[0]==0):
            O = np.kron(P0x0,I)
        elif(target_qubits[0]==1):
            O = np.kron(I,P0x0)
        else:
            O = np.kron(I,I)
        for i in range(2,total_qubits):
            if(i==target_qubits[0]):
                O = np.kron(O,P0x0)
            else:
                O = np.kron(O,I)

        o_control = O
        if(target_qubits[0]==0):
            O = P1x1
        elif(target_qubits[1]==0):
            O = X
        else:
            O = I
        for i in range(1,total_qubits):
            if(target_qubits[0]==i):
                O = np.kron(O,P1x1)
            elif(target_qubits[1]==i):
                O = np.kron(O,X)
            else:
                O = np.kron(O,I)
        o_target = O
        O = o_control + o_target

        return O


def run_program(total_qubits,initial_state, program):

    for i in range(len(program)):
        initial_state = np.dot(initial_state,get_operator(total_qubits,program[i]['gate'
],program[i]['target'],program[i]['params']))

    return initial_state


def get_counts(total_qubit,state_vector, num_shots):

    state = np.abs(state_vector)**2
    sample_List = []
    for i in range(len(state_vector)):
```

```
        sample_List.append(bin(i)[2:].zfill(total_qubit))

    randomNumberList = choice(
      sample_List, num_shots, p=state)

    unique, counts = np.unique(randomNumberList, return_counts=True)
    count = dict(zip(unique, counts))
    return count
```

In [139]:

```
# Circuit 1
# Angeles in Radians
my_circuit = [
{ "gate": "u3", "target": [0] , "params":{"theta":1.5708,"phi":1.5708,"lamb":1.5708}}
]



total_qubits = 2
my_qpu = get_ground_state(total_qubits)


# Run circuit

final_state = run_program(total_qubits,my_qpu, my_circuit)


# Read results

counts = get_counts(total_qubits,final_state, 1000)

print(counts)
```

```
{'00': 505, '10': 495}
```

In [140]:

```
# Circuit 2

my_circuit = [
{ "gate": "h", "target": [0] ,"params":0},
{ "gate": "cx", "target": [0,2] ,"params":0}
]
total_qubits = 3
my_qpu = get_ground_state(total_qubits)


# Run circuit

final_state = run_program(total_qubits,my_qpu, my_circuit)


# Read results

counts = get_counts(total_qubits,final_state, 1000)

print(counts)
```

```
{'000': 496, '101': 504}
```

In [141]:

```
# Circuit 3

my_circuit = [
{ "gate": "h", "target": [0] ,"params":0},
{ "gate": "cx", "target": [0, 1] ,"params":0},
    { "gate": "h", "target": [2],"params":0}
]
total_qubits = 5
```

```
my_qpu = get_ground_state(total_qubits)

# Run circuit
final_state = run_program(total_qubits,my_qpu, my_circuit)

# Read results
counts = get_counts(total_qubits,final_state, 1000)
print(counts)
```

```
{'00000': 235, '00100': 251, '11000': 237, '11100': 277}
```

In [ ]:

```

```