```
/tmp/ipykernel_110/488951255.py:27: DeprecationWarning: The package qiski
t.providers.ibmq is being deprecated. Please see https://ibm.biz/provider
_migration_guide to get instructions on how to migrate to qiskit-ibm-prov
ider (https://github.com/Qiskit/qiskit-ibm-provider) and qiskit-ibm-runti
me (https://github.com/Qiskit/qiskit-ibm-runtime).
  provider = IBMQ.load_account()
/tmp/ipykernel_110/488951255.py:27: DeprecationWarning: The qiskit.IBMQ e
ntrypoint and the qiskit-ibmq-provider package (accessible from 'qiskit.p
roviders.ibmq`) are deprecated and will be removed in a future release. I
nstead you should use the qiskit-ibm-provider package which is accessible
from 'qiskit_ibm_provider'. You can install it with 'pip install qiskit_i
bm_provider'. Just replace 'qiskit.IBMQ' with 'qiskit_ibm_provider.IBMPro
vider'
  provider = IBMQ.load_account()
/tmp/ipykernel_110/488951255.py:35: UserWarning: seed_simulator is not a
recognized runtime option and may be ignored by the backend.
  results = Backend_d.run(Transpiled_circuit,seed_simulator=42,shots=shot
s).result()
```

Task set 2, QFT density matrices

In [5]:

```python
# Ex 1
from qiskit import Aer, transpile, assemble
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit.quantum_info import Statevector
from qiskit.visualization import plot_bloch_multivector
import numpy as np
from qiskit.visualization import plot_histogram

q = QuantumRegister(3) # Qubit dimension definition
qc = QuantumCircuit(q)
#-------------------------------------------------------------------------
# qc.x(q[0])

qc.ry(2*np.arccos(1/(np.sqrt(3))), q[0])
qc.ch(q[0], q[1])
qc.cx(q[1], q[2])
qc.cx(q[0], q[1])
qc.x(q[0])

qc.draw('mpl')
#-------------------------------------------------------------------------
# State vector?

A=Statevector(qc)
print(np.array(A))
```

```
[0.        +0.j 0.57735027+0.j 0.57735027+0.j 0.        +0.j
 0.57735027+0.j 0.        +0.j 0.        +0.j 0.        +0.j]
```

In [17]:
```
# Ex 2
import numpy as np

from qiskit.quantum_info import DensityMatrix


q = DensityMatrix.from_instruction(qc)

# Hermitian
# assert(q == np.conj(q).T)

# Positive
assert(np.all(np.linalg.eigvals(q) > 0))
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call las
Cell In[17], line 12
      6 q = DensityMatrix.from_instruction(qc)
      8 # Hermitian
      9 # assert(q == np.conj(q).T)
     10
     11 # Positive
---> 12 assert(np.all(np.linalg.eigvals(q) > 0))

AssertionError:
```

In [31]:
```
# Ex 3
from qiskit.quantum_info import random_unitary

randomU = random_unitary(8)

print(q)

q_new = randomU @ q @ np.conj(randomU).T
ex_lambda = np.trace(q_new @ q_new)

assert(ex_lambda <= 1)
print(ex_lambda)
```

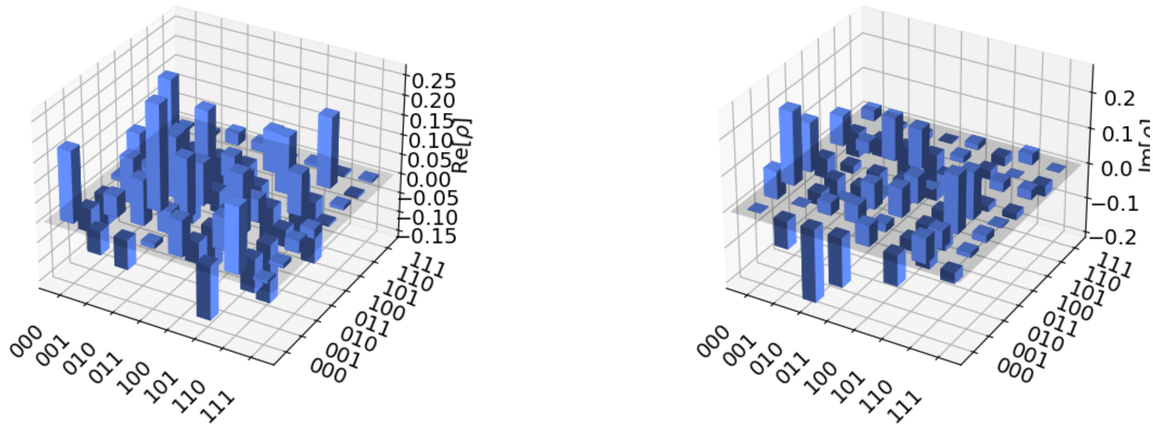
```
DensityMatrix([[0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j],
               [0.        +0.j, 0.33333333+0.j, 0.33333333+0.j,
                0.        +0.j, 0.33333333+0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j],
               [0.        +0.j, 0.33333333+0.j, 0.33333333+0.j,
                0.        +0.j, 0.33333333+0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j],
               [0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j],
               [0.        +0.j, 0.33333333+0.j, 0.33333333+0.j,
                0.        +0.j, 0.33333333+0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j],
               [0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j],
               [0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j],
               [0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j, 0.        +0.j,
                0.        +0.j, 0.        +0.j]],
              dims=(2, 2, 2))
(0.9999999999999994+1.098088285171487e-18j)
```

In [33]:

```python
# Ex 4
from qiskit.visualization import plot_state_city

plot_state_city(q_new)
```

Out[33]:



In [43]:

```python
# Ex 5
from qiskit.quantum_info import partial_trace
import qiskit
q_ab = qiskit.quantum_info.random_density_matrix(4)

print(q_ab)

q_a = partial_trace(q_ab,[1])

print(q_a)
```

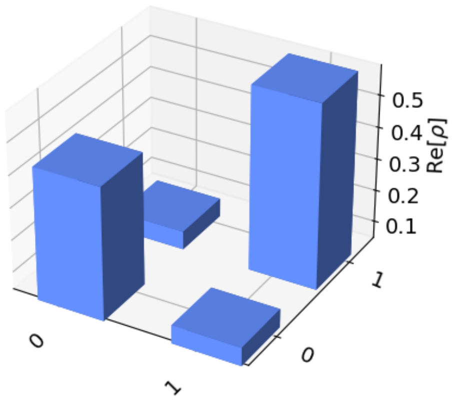
```
DensityMatrix([[ 0.15976714+0.j        , -0.03127317+0.04532782j,
                  0.0429686 +0.03942619j,  0.10415908-0.07239393j],
                [-0.03127317-0.04532782j,  0.3181747 +0.j        ,
                  0.02083999-0.01912208j, -0.09793156-0.02101915j],
                [ 0.0429686 -0.03942619j,  0.02083999+0.01912208j,
                  0.25850887+0.j        ,  0.088778  +0.07825897j],
                [ 0.10415908+0.07239393j, -0.09793156+0.02101915j,
                  0.088778  -0.07825897j,  0.26354929+0.j        ]],
               dims=(2, 2))
DensityMatrix([[0.41827601+0.j        , 0.05750483+0.12358679j],
                [0.05750483-0.12358679j, 0.58172399+0.j        ]],
               dims=(2,))
```

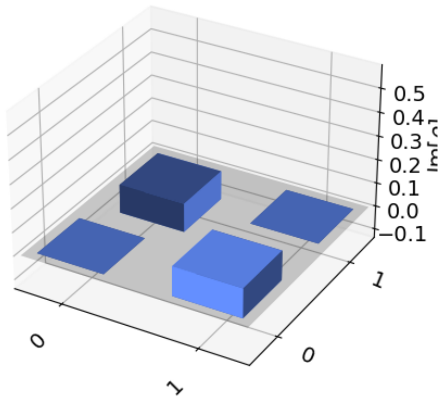
In [44]:

```
# Ex 6
from qiskit.visualization import plot_state_city

plot_state_city(q_a)
```

Out[44]:





In [52]:
```
# Quantum Fourier transform
# Ex 1
# |b> = QFT|1010>
# import numpy.pi as pi

def qft_rotations(circuit, n):
    if n == 0: # Exit function if circuit is empty
        return circuit
    n -= 1 # Indexes start from 0
    circuit.h(n) # Apply the H-gate to the most significant qubit
    for qubit in range(n):
        # For each less significant qubit, we need to do a
        # smaller-angled controlled rotation:
        circuit.cp(np.pi/2**(n-qubit), qubit, n)

def swap_registers(circuit, n):
    for qubit in range(n//2):
        circuit.swap(qubit, n-qubit-1)
    return circuit

def qft(circuit, n):
    """QFT on the first n qubits in circuit"""
    qft_rotations(circuit, n)
    swap_registers(circuit, n)
    return circuit

# Let's see how it looks:
qc = QuantumCircuit(4)
# preparing the initial state
qc.x(0)
qc.x(2)
# applying qft
qft(qc,4)
qc.draw()
```

Out[52]: