# Qiskit

# Inclusão de bibliotecas

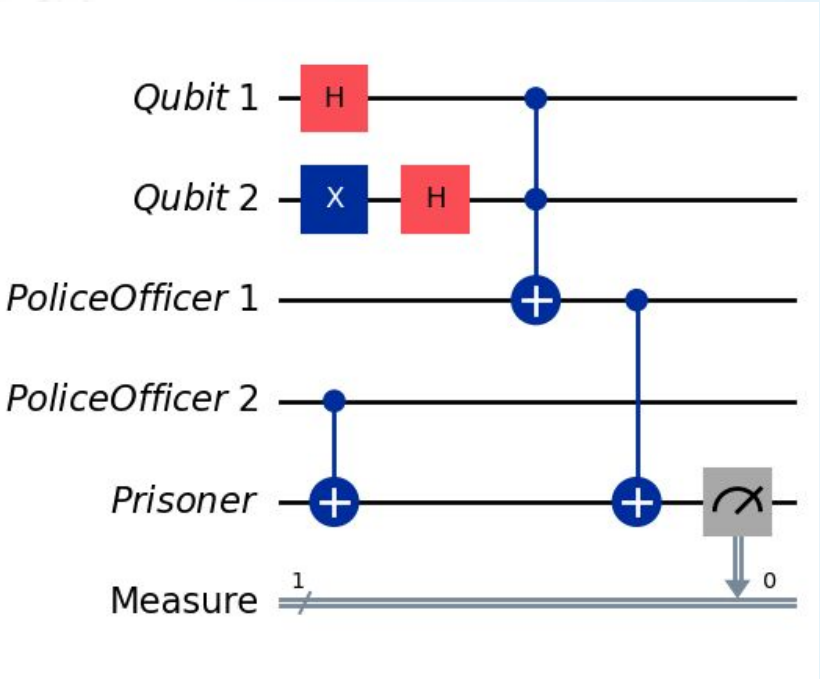
```
!pip install qiskit
!pip install qiskit-aer
```

```
from qiskit import *
from  qiskit_aer import Aer
from qiskit.visualization import
plot_histogram,
plot_bloch_multivector
import numpy
import qiskit
```

UFPEL

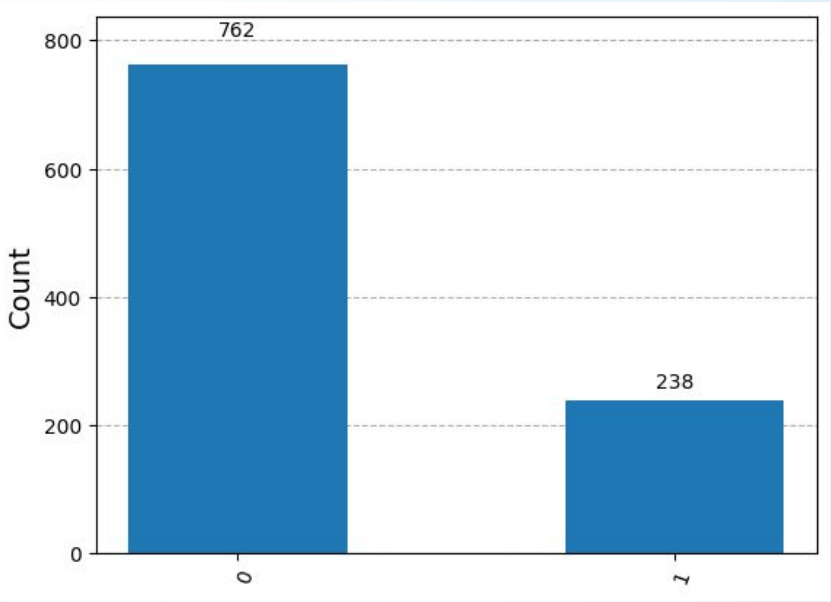
# Construção de circuito

```
Qubit1 = QuantumRegister(1, 'Qubit 1')
Qubit2 = QuantumRegister(1, 'Qubit 2')
Police1 = QuantumRegister(1, 'PoliceOfficer 1')
Police2 = QuantumRegister(1, 'PoliceOfficer 2')
Prisoner = QuantumRegister(1, 'Prisoner')
creg_c = ClassicalRegister(1, 'Measure') # quantidade de bits clássicos

# QuantumCircuit(nro bits quânticos, nro bits clássicos)
# FCircuit2 = QuantumCircuit(5, 1)
FCircuit2 = QuantumCircuit(Qubit1, Qubit2, Police1, Police2, Prisoner, creg_c)

# FCircuit2.h(0)
# FCircuit2.x(1)
# FCircuit2.h(1)

FCircuit2.h(Qubit1)
FCircuit2.x(Qubit2)
FCircuit2.h(Qubit2)

# FCircuit2.ccx(0, 1, 2)
FCircuit2.ccx(Qubit1, Qubit2, Police1)

# FCircuit2.cx(3,4)
# FCircuit2.cx(2,4)
FCircuit2.cx(Police2, Prisoner)
FCircuit2.cx(Police1, Prisoner)

# measure(bit quântico, bit clássico)
# FCircuit2.measure(4, 0)
FCircuit2.measure(Prisoner, 0)
FCircuit2.draw(output='mpl')
```

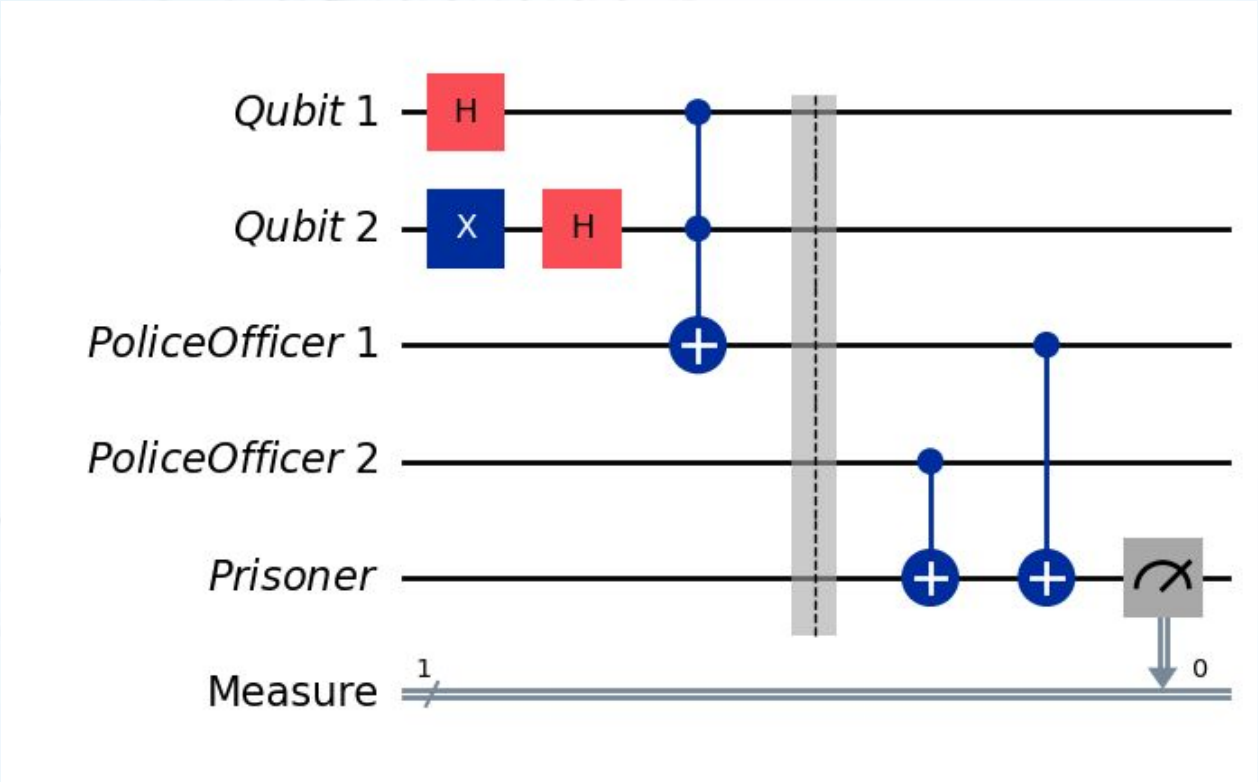



UFPEL

```
sim = Aer.get_backend('aer_simulator')
result = sim.run(FCircuit2, shots = 1000).result()
counts = result.get_counts()
print(counts)
plot_histogram(counts)
```

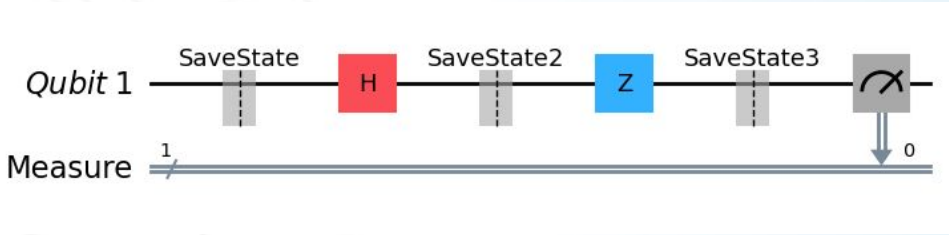



UFPEL


Qubit 1
H
Qubit 2
X
H
PoliceOfficer 1
PoliceOfficer 2
Prisoner
Measure
1
0

# Save statevector

```
Qubit1 = QuantumRegister(1, 'Qubit 1')
creg_c = ClassicalRegister(1, 'Measure')
circuit2 = QuantumCircuit(Qubit1, creg_c)
circuit2.save_statevector(label='SaveState')
circuit2.h(Qubit1)
circuit2.save_statevector(label='SaveState2')
circuit2.z(Qubit1)
circuit2.save_statevector(label='SaveState3')
circuit2.measure(Qubit1, 0) # salva no qubit 0
circuit2.draw(output='mpl')
```



```
backend = Aer.get_backend("statevector_simulator")
result = sim.run(circuit2, backend = backend, shots =
1000).result()
print(result.data(0)['SaveState'])
```

```
Statevector([1.+0.j, 0.+0.j],dims=(2,))
```

```
backend = Aer.get_backend("statevector_simulator")
result = sim.run(circuit2, backend = backend, shots =
1000).result()
print(result.data(0)['SaveState2'])
```

```
Statevector([0.70710678+0.j,
0.70710678+0.j],dims=(2,))
```

```
backend = Aer.get_backend("statevector_simulator")
result = sim.run(circuit2, backend = backend, shots =
1000).result()
print(result.data(0)['SaveState3'])
```

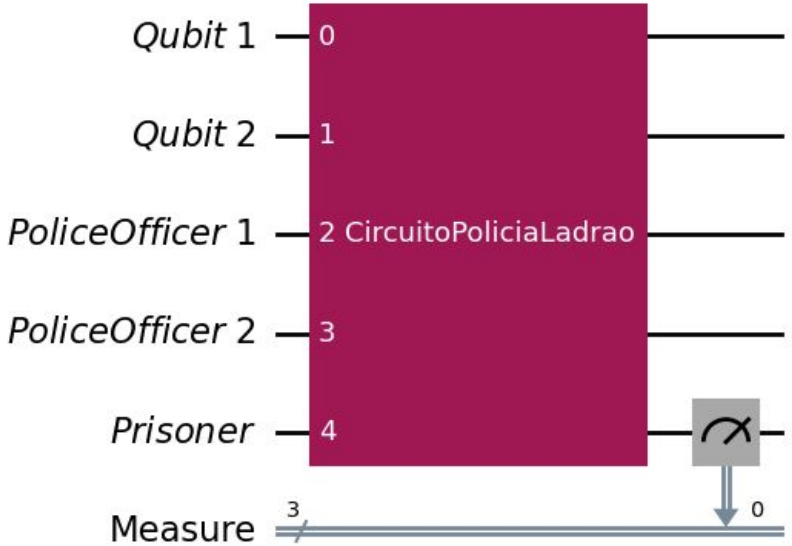
```
Statevector([ 0.70710678+0.j,
-0.70710678+0.j],dims=(2,))
```

UFPEL

```python
def BlackBox(*args):
  circuitParam = QuantumCircuit(*args)
  circuitParam.h(Qubit1)
  circuitParam.x(Qubit2)
  circuitParam.h(Qubit2)
  circuitParam.ccx(Qubit1, Qubit2, Police1)
  circuitParam.cx(Police2, Prisoner)
  circuitParam.cx(Police1, Prisoner)
  ParamGate = circuitParam.to_gate()
  ParamGate.name = "CircuitoPoliciaLadrao"
  return ParamGate
```

# Caixa preta

```python
Qubit1 = QuantumRegister(1, 'Qubit 1')
Qubit2 = QuantumRegister(1, 'Qubit 2')
Police1 = QuantumRegister(1, 'PoliceOfficer 1')
Police2 = QuantumRegister(1, 'PoliceOfficer 2')
Prisoner = QuantumRegister(1, 'Prisoner')
creg_c = ClassicalRegister(3, 'Measure')

circuit2 = QuantumCircuit(Qubit1, Qubit2, Police1, Police2, Prisoner, creg_c)

circReturn = BlackBox(Qubit1, Qubit2, Police1, Police2, Prisoner)


circuit2.append(circReturn, [0,1,2,3,4])

circuit2.measure(4, creg_c[0])
circuit2.draw(output='mpl')
```

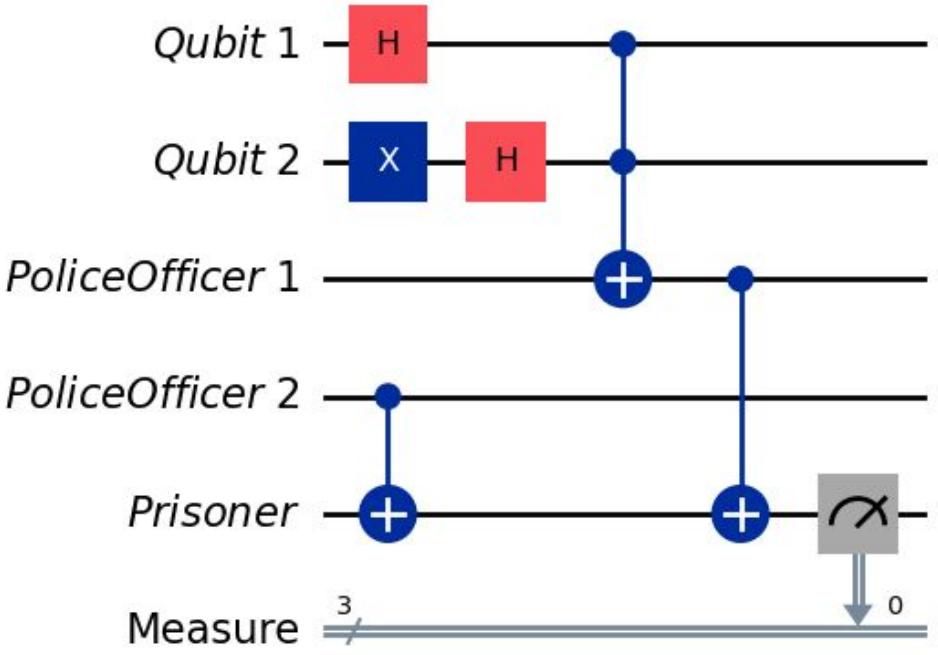



UFPEL

# Decompor caixa preta

```
circuit2.decompose().draw(output='mpl')
```

```
sim = Aer.get_backend('aer_simulator')
result = sim.run(circuit2.decompose(), shots =
1000).result()
counts = result.get_counts()
print(counts)
plot_histogram(counts)
```



# Custom Gates

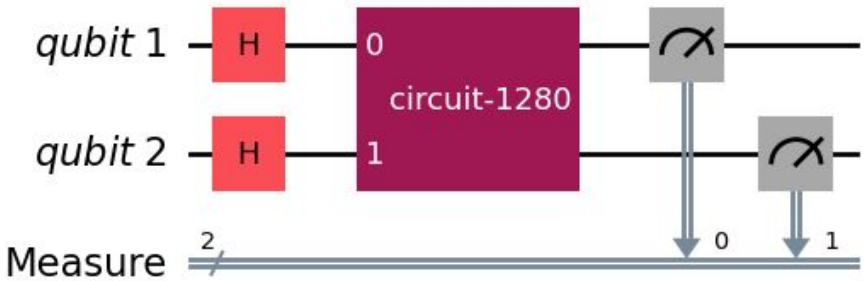
```
def CustomCNOT():
  circ = QuantumCircuit(2)
  circ.append(CustomGATE().control(1),[0,1])
  return circ

def CustomGATE():
  matrix = [[0,1],[1,0]]
  xGate = qiskit.circuit.library.UnitaryGate(matrix)
  return xGate
```

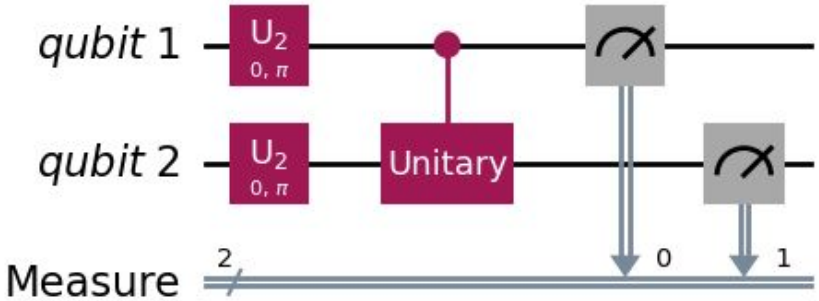
```
qubit1 = QuantumRegister(1, 'qubit 1')
qubit2 = QuantumRegister(1, 'qubit 2')
creg_c = ClassicalRegister(2, 'Measure')

circuit3 = QuantumCircuit(qubit1, qubit2,
creg_c)

circuit3.h(0)
circuit3.h(1)
returnControled = CustomCNOT()

circuit3.append(returnControled, [0,1])

circuit3.measure(0, 0)
circuit3.measure(1, 1)

circuit3.draw(output='mpl')
```

UFPEL

# Custom Gates

```
circuit3_decomposed = circuit3.decompose()
circuit3_decomposed.draw(output='mpl')
```




UFPEL