QKRISHI IBM Developer Certification Course

IBM CERTIFIED ASSOCIATE DEVELOPER - QUANTUM COMPUTATION USING QISKIT V0.2X

- Learn the fundamentals of quantum computing and Qiskit
- Gain hands-on experience with quantum algorithms and quantum programming
- This course will cover the fundamentals of quantum computing and the Qiskit framework
- By the end of this course, you will be prepared to take the IBM Certified Associate Developer - Quantum Computation using Qiskit v0.2X exam

Learning Material Developed by: Padmapriya Mohan and Vaishnavi Markunde

Section 1

Introduction to Quantum Circuits

Section 1 introduces the foundations of qiskit. The topics covered are -

1.Constructing quantum circuits
2.Single-qubit gates
3.Measurement
4.Circuit Depth

In [1]:

```
## imports


from qiskit import *
from qiskit.visualization import plot_histogram
import numpy as np
```

Creating a Quantum Circuit

A quantum circuit initializes a qubit in the state $|0\rangle$

In [2]:

```
# defining number of qubits
n = 2
```

In [3]:

```
# creating a circuit with n qubits
qc = QuantumCircuit(n)

# circuit visualization
qc.draw()
```

Out[3]:

```
q_0:

q_1:
```

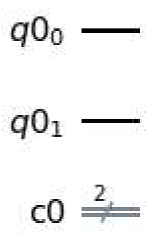
In [4]:

```
## Another way of creating a circuit using quantum and classical registers
qr = QuantumRegister(n)
cr = ClassicalRegister(n)
qc = QuantumCircuit(qr, cr)
```

In [5]:

```
# using matplotlib library to generate the visualization of the circuit
qc.draw('mpl')
```

Out[5]:



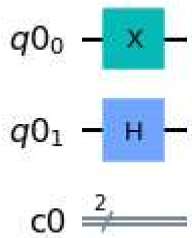
Applying Single qubit gates

In Qiskit, single-qubit gates can be applied on qubits in a quantum circuit using the methods provided by the **QuantumCircuit** object. The names of these methods correspond to the names of the gates they apply.
For example, the h() method applies the Hadamard gate, and the x() method applies the Pauli-X (NOT) gate.
Qubit index (0-based indexing) is passed as an argument to the gates on which the gate has to be applied

In [6]:

```
qc.x(0)
qc.h(1)
qc.draw('mpl')
```

Out[6]:



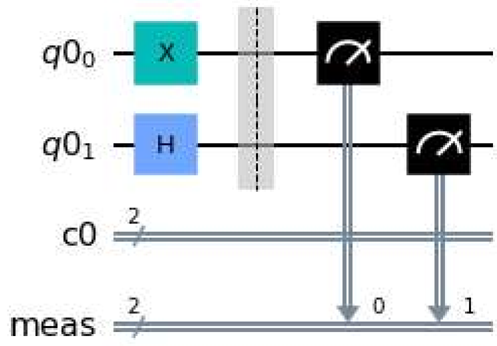
Measuring the circuit

Qubit measurement outcomes are stored in corresponding classical bits

In [7]:

```
qc.measure_all()
qc.draw('mpl')
```

Out[7]:



The operator after the gates is called 'barrier' and can be used to seperate various gates. The above function 'measure_all' measures all the qubits and stores in the results in corresponding classical bits.

Another way to measure qubits is to use the measure() method, which allows you to specify the qubits to be measured and the classical register where the measurement outcomes will be stored.
Here's the example:

In [8]:

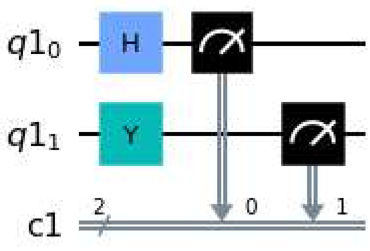
```
qr = QuantumRegister(n)
cr = ClassicalRegister(n)
qc = QuantumCircuit(qr, cr)

# Apply a Hadamard gate to the first qubit
qc.h(qr[0])

# Apply a Y-gate to the second qubit
qc.y(qr[1])
```

```
# Measure the first qubit
qc.measure(qr[0], cr[0])
# Measure the second qubit
qc.measure(qr[1],cr[1])
qc.draw('mpl')
```

Out[8]:



Alternatively , you can measure multiple qubits at once using a list

In [9]:

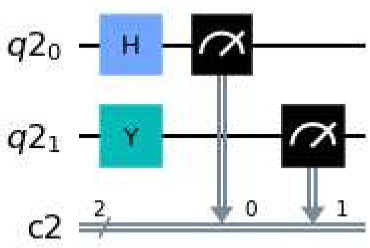
```
#Creating n qubit quantum register
qr = QuantumRegister(n)
cr = ClassicalRegister(n)
qc = QuantumCircuit(qr, cr)

# Apply a Hadamard gate to the first qubit
qc.h(qr[0])

# Apply a Y-gate to the second qubit
qc.y(qr[1])

# Measure multiple qubits, first list corresponds to qubits and the second list corresponds to classical bits
qc.measure([0,1],[0,1])
qc.draw('mpl')
```

Out[9]:



Circuit Depth

Circuit depth refers to the number of quantum gates in a quantum circuit that are applied to a qubit before a measurement is made. It is a measure of the complexity of the quantum circuit. To return the circuit depth of a circuit in Qiskit, you can use the depth() method. This method takes a QuantumCircuit object as input and returns the circuit depth as an integer.

Note: Barrier operation is not counted

In [10]:

```
# Create a quantum circuit with 2 qubits
qc = QuantumCircuit(2)

# Apply a Hadamard gate to the first qubit
qc.x(0)

# Apply a CNOT gate with control on the first qubit and target on the second qubit
qc.cx(0, 1)
qc.barrier()
qc.x(0)
print(qc)

# Get the circuit depth
depth = qc.depth()

# Print the circuit depth
print("Depth: ",depth)
```

```
     ┌───┐      ░ ┌───┐
q_0: ┤ X ├──■───░─┤ X ├
     └───┘┌─┴─┐ ░ └───┘
q_1: ─────┤ X ├─░──────
          └───┘ ░
```

```
        └──┘ ░
Depth:  3
```

Different single qubit gates

There are several single-qubit gates that are commonly used in quantum computing and are supported in Qiskit.
Here is a list of some of the most commonly used single-qubit gates

Pauli-X gate

This gate is represented by the following matrix, and it flips the state of a qubit, i.e., it maps |0⟩ to |1⟩ and |1⟩ to |0⟩. In Qiskit, the Pauli-X gate can be applied to a qubit using the **`x()`** method of the `QuantumCircuit` object.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Pauli-Y gate

This gate is represented by the following matrix, and it applies a phase of -i to the state of a qubit, i.e., it maps |0⟩ to i|1⟩ and |1⟩ to -i|0⟩. In Qiskit, the Pauli-Y gate can be applied to a qubit using the **`y()`** method of the `QuantumCircuit` object.

$$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Pauli-Z gate

This gate is represented by the following matrix, and it applies a phase of -1 to the |1⟩ state of a qubit,i.e., it maps |0⟩ to |0⟩ and |1⟩ to -|1⟩. In Qiskit, the Pauli-Z gate can be applied to a qubit using the **`z()`** method of the `QuantumCircuit` object.

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Hadamard gate (H)

The Hadamard gate maps the state |0⟩ to $\frac{1}{\sqrt{2}}$(|0⟩ + |1⟩) and |1⟩ to $\frac{1}{\sqrt{2}}$(|0⟩ - |1⟩). it's often used for quantum state preparation and superposition. In Qiskit, the Hadamard gate can be applied to a qubit using the **`h()`** method of the `QuantumCircuit` object.

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

S- gate

It applies a phase shift of pi/2 radians to the state |0⟩, it maps |0⟩ to |0⟩ and |1⟩ to i|1⟩. In Qiskit, the S gate can be applied to a qubit using the **`s()`** method of the `QuantumCircuit` object.

$$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

T-gate

This gate is represented by the following matrix and applies a phase of pi/4 to the |1⟩ state of a qubit. It maps |0⟩ to |0⟩ and |1⟩ to $e^{i\frac{\pi}{4}}$. It is often used together with the Hadamard gate to make a phase estimation algorithm. In Qiskit, the Pi/8 gate can be applied to a qubit using the **`t()`** method of the `QuantumCircuit` object.

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$$

Phase Gate

The phase shift gate applies an arbitrary phase shift to the qubit state. In Qiskit, this gate can be applied to a qubit using the **`u1(λ)`** method of the `QuantumCircuit` object., where λ is the angle of the phase shift in radians.

$$\begin{bmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

Rotation gates:

There are three rotation gates, rx(), ry() and rz() each representing rotation around x,y,z axis respectively. The **`rx()`** method is used to apply a rotation of angle around the x-axis, **`ry()`**) around y-axis and **`rz()`** around z-axis.

> Rx gate with pi/2:

$$\begin{bmatrix} \cos(\frac{\pi}{4}) & -i\sin(\frac{\pi}{4}) \\ -i\sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{bmatrix}$$

> Ry gate with pi/2:

$$\begin{bmatrix} \cos(\frac{\pi}{4}) & -\sin(\frac{\pi}{4}) \\ \sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{bmatrix}$$

> Rz gate with pi/2:

$$\begin{bmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$$