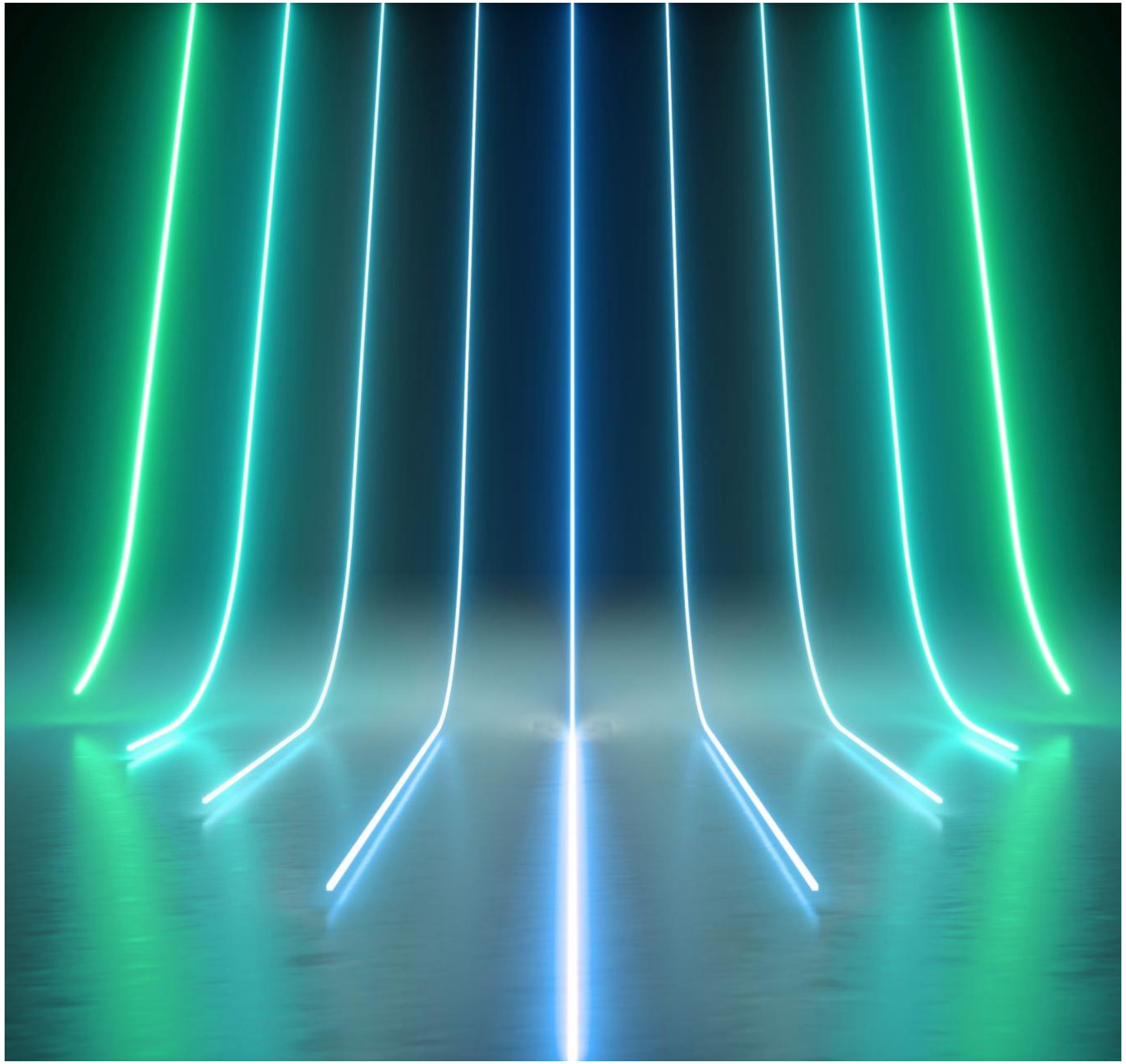
Fundamentals of Qiskit v1.0 and Quantum Circuit Design

Seonggeun Park

Korea University



Seonggeun Park

ssgg0926@korea.ac.kr

Undergraduate student at Korea Univ.

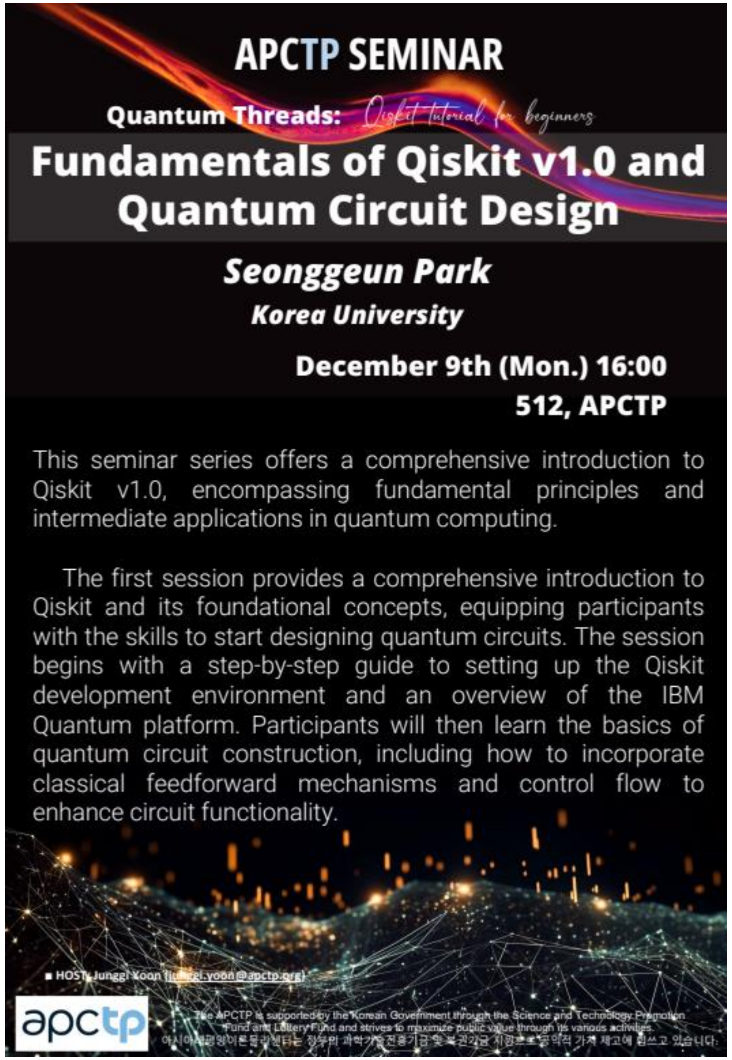
The president of QUICK

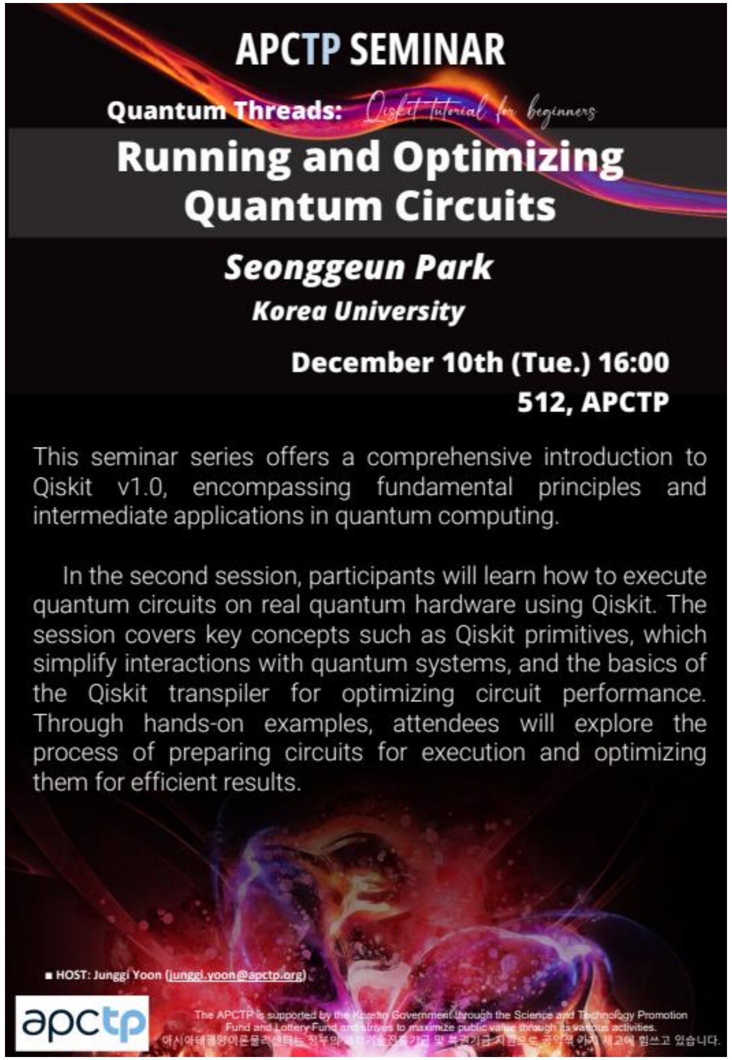
Center Director Prize, 2024 Quantum Information Competition

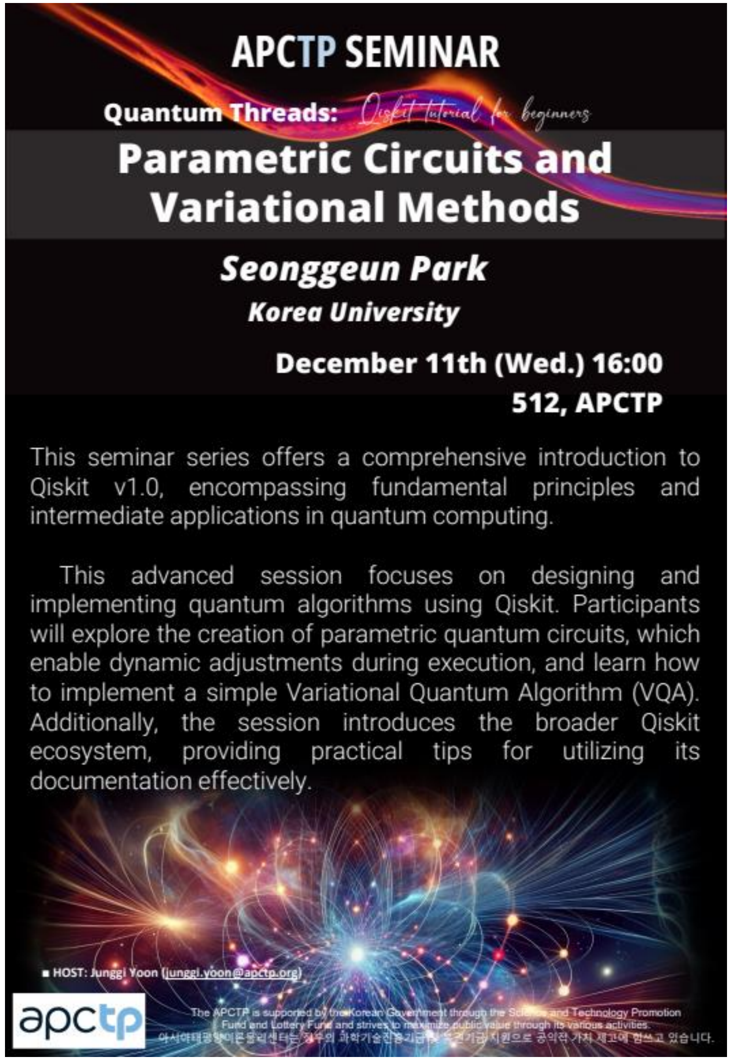
1st Place, MIT iQuHACK 2024, IonQ remote challenge division

9th Place, QHack 2024 coding challenge

Qiskit Tutorial for Beginner Series at APCTP







Qiskit Tutorial for Beginner Series at APCTP

- Overview of Basic Quantum Information
- Introduction to Qiskit & IBM Quantum Platform
- Setting Up the Qiskit Development Environment
- Designing Quantum Circuits
- Implementing Classical Feedforward and Control Flow

- Exploring Qiskit Runtime Service
- Executing Quantum Circuits using Qiskit Primitives
- Transpiling Quantum Circuits

- Implementing Parametric Quantum Circuits & VQA
- Browsing the Qiskit Ecosystem

Qiskit Tutorial for Beginner Series at APCTP

- Overview of Basic Quantum Information
- Introduction to Qiskit & IBM Quantum Platform
- Setting Up the Qiskit Development Environment
- Designing Quantum Circuits
- Implementing Classical Feedforward and Control Flow

- Exploring Qiskit Runtime Service
- Executing Quantum Circuits using Qiskit Primitives
- Transpiling Quantum Circuits

- Implementing Parametric Quantum Circuits & VQA
- Browsing the Qiskit Ecosystem

Contents

I.
Overview of Basic Quantum Information

Quantum States

- The state of the particle is represented by a vector $|\psi(t)\rangle$ in a Hilbert space.
- Basis: $\{|0\rangle, |1\rangle, \cdots, |d-1\rangle\}$, $\langle i|j\rangle = \delta_{ij}$ for $\forall i, j \in \{0, 1, \cdots, d-1\}$

  Quantum state: $|\psi\rangle = \sum_0^{d-1} \alpha_i |i\rangle$

  Dual vector: $\langle\psi| = \sum_0^{d-1} \alpha_i^* \langle i|$

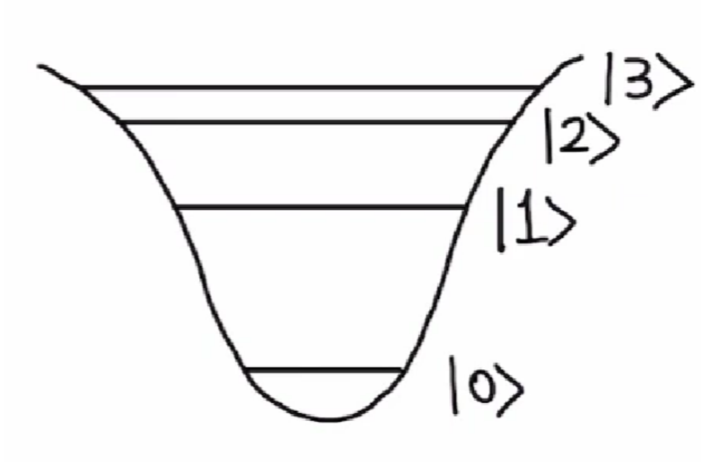
  Normalization: $\langle\psi|\psi\rangle = 1$

Qubit, Qutrit, Qudit

Qubit: 2-level system

Qutrit: 3-level system

Qudit: d-level system



Qubit

$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

$\langle\psi|\psi\rangle = 1 \;\; \rightarrow \;\; \alpha\alpha^* + \beta\beta^* = |\alpha|^2 + |\beta|^2 = 1$

Measuring $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in the Z basis,

the probability of observing $|0\rangle$ is $|\alpha|^2$, and

the probability of observing $|1\rangle$ is $|\beta|^2$.

Qubit States

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Z basis

$$|+\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|-\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

X basis

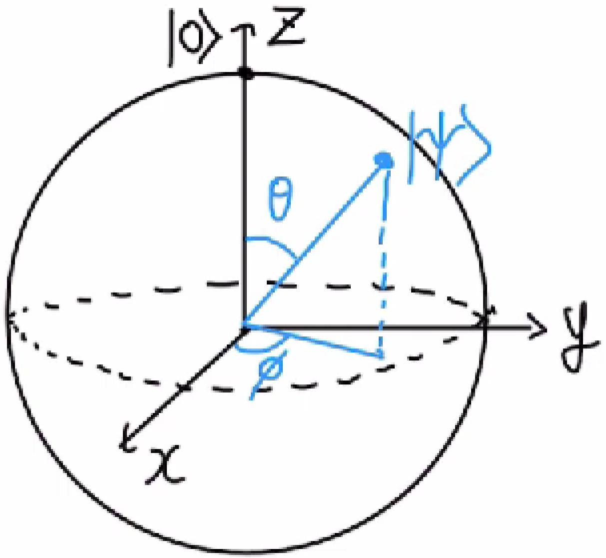
$$|+i\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ i \end{bmatrix} \qquad = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$$

$$|-i\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -i \end{bmatrix} \qquad = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$$

Y basis

Bloch Sphere

γ: global phase



$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right)$$

Bloch Sphere

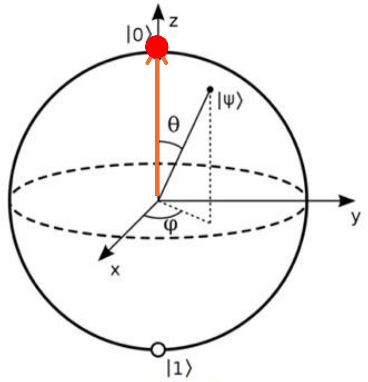
$$|\psi>= \cos\frac{\theta}{2}|0>+e^{i\phi}\sin\frac{\theta}{2}|1>$$

$$|+>= \frac{1}{\sqrt{2}}|0>+\frac{1}{\sqrt{2}}|1>$$

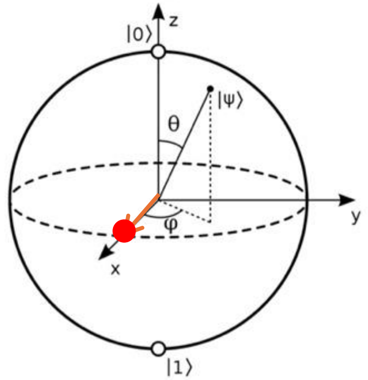
$$|->= \frac{1}{\sqrt{2}}|0>-\frac{1}{\sqrt{2}}|1>$$

$$|+i>= \frac{1}{\sqrt{2}}|0>+\frac{i}{\sqrt{2}}|1>$$

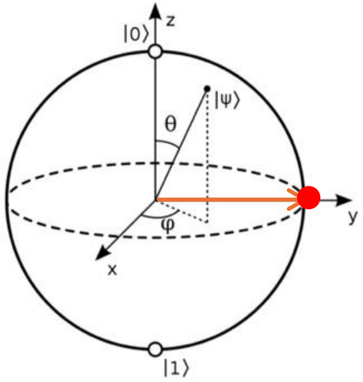
$$|-i>= \frac{1}{\sqrt{2}}|0>-\frac{i}{\sqrt{2}}|1>$$



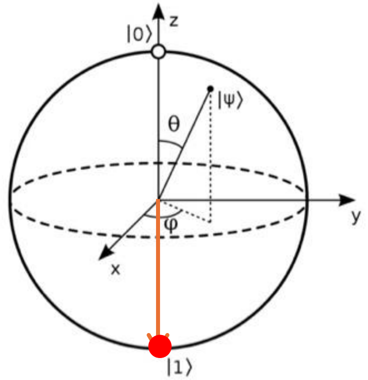
$$|0>= \begin{bmatrix}1\\0\end{bmatrix}$$



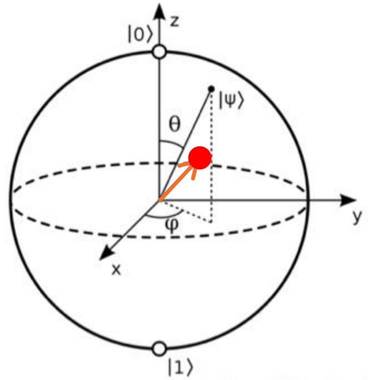
$$|+>= \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$$



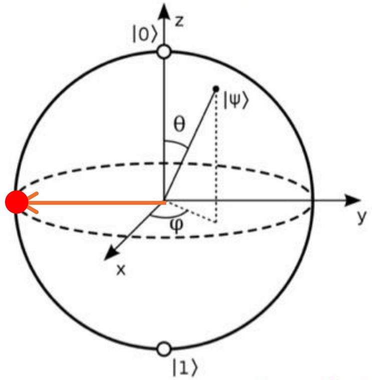
$$|+i>= \frac{1}{\sqrt{2}}\begin{bmatrix}1\\i\end{bmatrix}$$



$$|1>= \begin{bmatrix}0\\1\end{bmatrix}$$



$$|->= \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$$



$$|-i>= \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-i\end{bmatrix}$$

Single Qubit Gates

Quantum gates are Unitary operator

- Pauli X gate: $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

- Pauli Y gate: $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$

- Pauli Z gate: $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

- S gate: $S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$

- T gate: $\mathrm{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$

- Hadamard gate: $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

- RX gate: $RX = e^{-i\frac{\theta}{2}X} = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$

- RY gate: $RY = e^{-i\frac{\theta}{2}Y} = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$

- RZ gate: $RZ = e^{-i\frac{\theta}{2}Z} = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$

Single Qubit Gates

$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ $|+\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$ $|+i\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\i\end{bmatrix}$

$|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$ $|-\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$ $|-i\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-i\end{bmatrix}$

$X = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$ $Y = \begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$ $Z = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$ $S = \begin{bmatrix}1 & 0\\0 & i\end{bmatrix}$

$\mathrm{T} = \begin{bmatrix}1 & 0\\0 & e^{i\frac{\pi}{4}}\end{bmatrix}$ $H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$

$$X|0\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$$

$$X|1\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$$

$$Z|0\rangle = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$$

$$Z|1\rangle = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\-1\end{bmatrix} = -|1\rangle$$

$$Z|+\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = |-\rangle$$

$$Z|-\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = |+\rangle$$

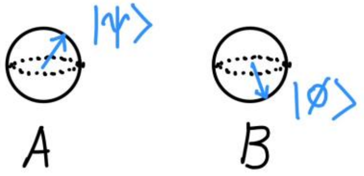
$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = |+\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = |-\rangle$$

$$Z = S^2 = T^4$$

Multi-qubit Systems

- Tensor product



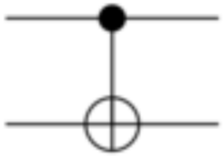
$|\psi\rangle_A$ and $|\phi\rangle_B \rightarrow |\Psi\rangle = |\psi\rangle_A \otimes |\phi\rangle_B$

- $\begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} c \\ d \end{bmatrix} \\ b \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}$

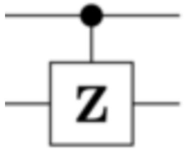
- $X_A \otimes I_B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & 1 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ 1 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & 0 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

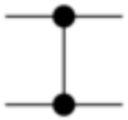
Multi-qubit Gates

- CNOT gate: $CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

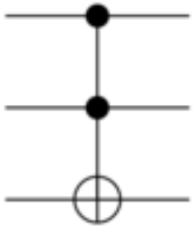


- CZ gate: $CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$



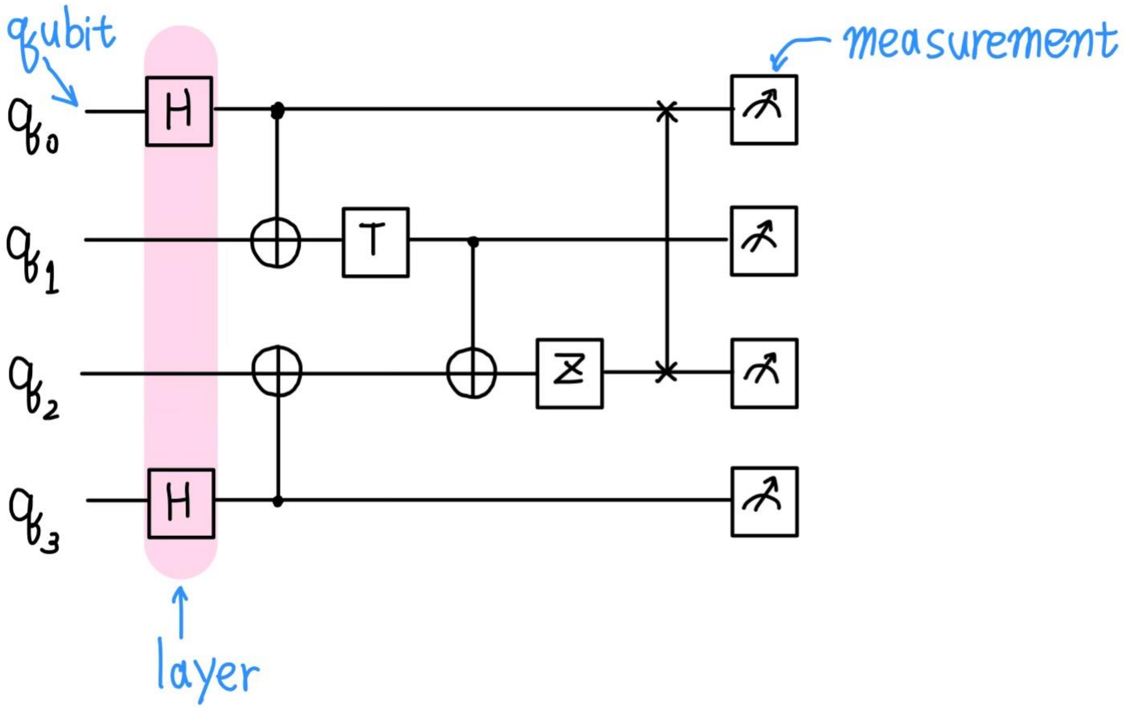


- Toffoli gate: $CCX = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$



Quantum Circuit

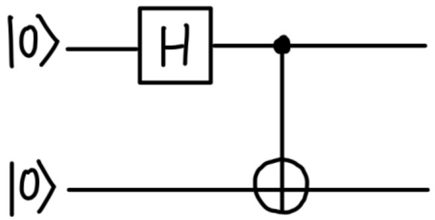
- A quantum circuit is a model of quantum computation where a computation consists of a sequence of qubit initializations, quantum gates, and measurements.



Quantum Circuit

- A quantum circuit is a model of quantum computation where a computation consists of a sequence of qubit initializations, quantum gates, and measurements.

- Example: $CNOT(H_A \otimes I_B)(|0\rangle_A \otimes |0\rangle_B)$



$$
\begin{aligned}
CNOT(H_A \otimes I_B)(|0\rangle_A \otimes |0\rangle_B) &= CNOT(H_A|0\rangle_A \otimes I_B|0\rangle_B) \\
&= CNOT(\frac{|0\rangle_A + |1\rangle_B}{\sqrt{2}} \otimes |0\rangle_B) \\
&= \frac{1}{\sqrt{2}} CNOT(|0\rangle_A|0\rangle_B + |1\rangle_A|0\rangle_B) \\
&= \tfrac{1}{\sqrt{2}}(CNOT(|0\rangle_A|0\rangle_B) + CNOT(|1\rangle_A|0\rangle_B)) \\
&= \frac{1}{\sqrt{2}}(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)
\end{aligned}
$$

II.
Introduction to Qiskit & IBM Quantum Platform

Qiskit

Qiskit SDK is an open-source SDK for working with quantum computers at the level of quantum circuits, operators, and primitives.

https://github.com/Qiskit/qiskit

https://docs.quantum-computing.ibm.com/

Version 1.0 released Feb 2024

Qiskit Patterns is a framework for breaking down domain-specific problems into stage.

Qiskit

Qiskit Runtime is a cloud-based service for executing quantum computations on IBM Quantum hardware.

https://github.com/Qiskit/qiskit-ibm-runtime

https://docs.quantum-computing.ibm.com/

Runtime Primitives version 2

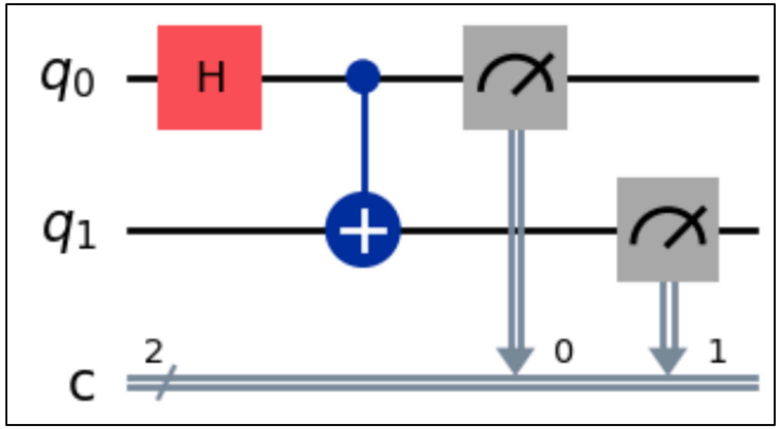
Released March 2024

The Qiskit Ecosystem is a collection of software and projects that build on or extend Qiskit

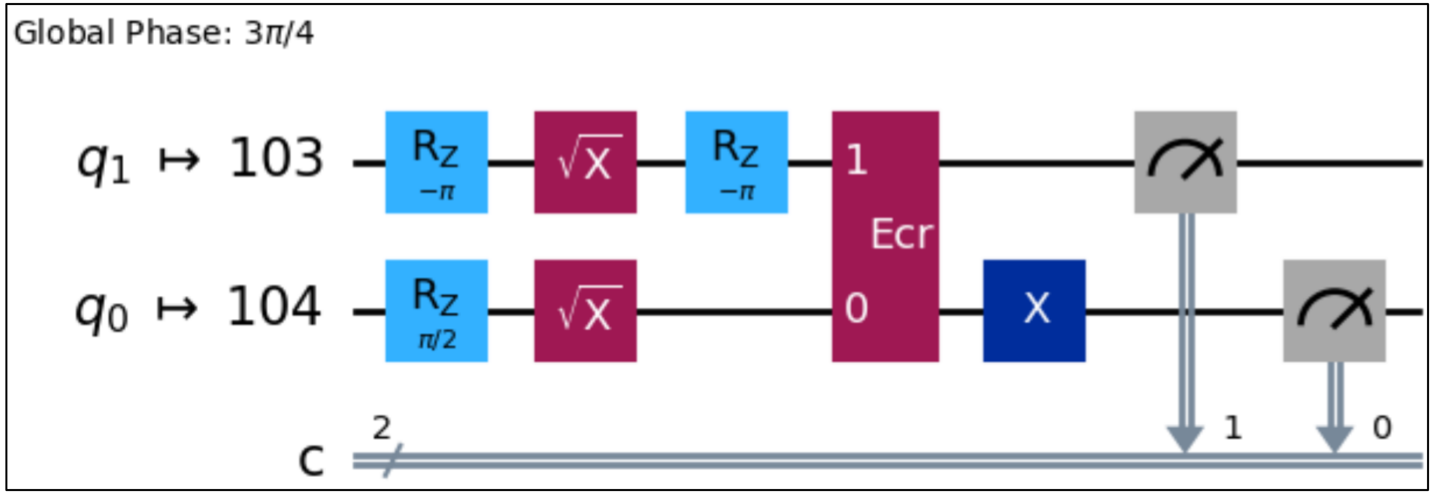
http://qiskit.github.io/ecosystem

source: Qiskit Global Summer School Lecture 1

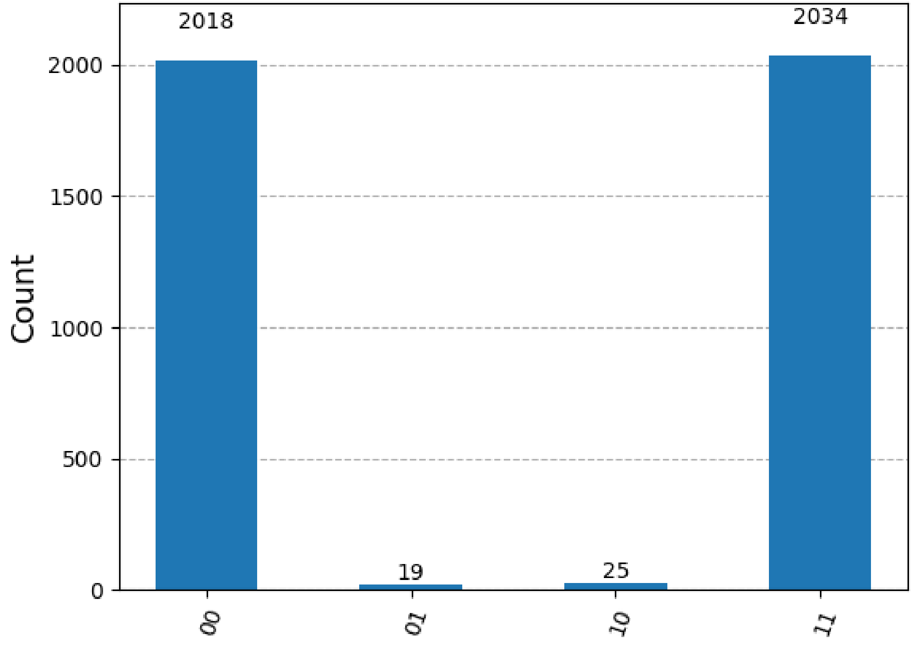
Qiskit



q_0
q_1
H
c
2
0
1



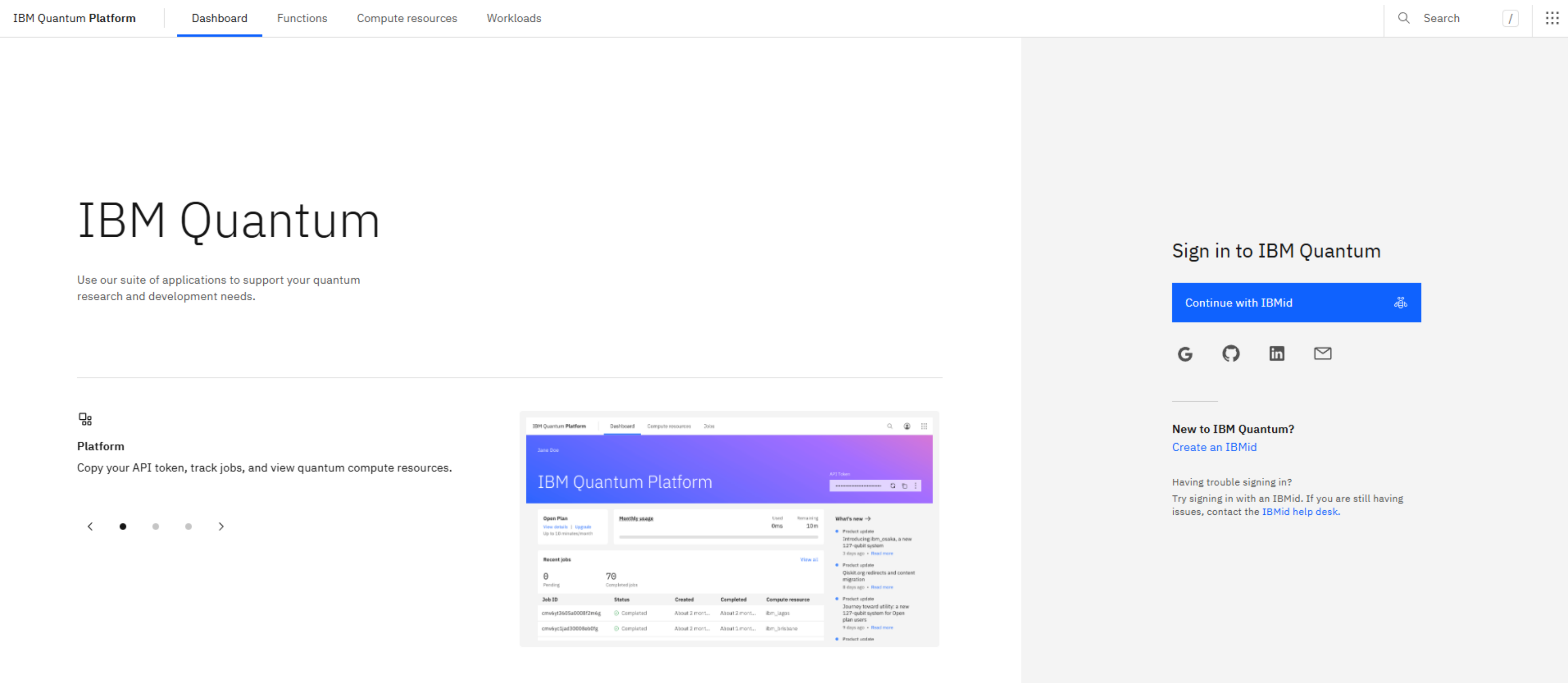
Global Phase: 3π/4
$q_1 \mapsto 103$
$q_0 \mapsto 104$
R_Z $-\pi$
$\sqrt{X}$
R_Z $-\pi$
R_Z $\pi/2$
$\sqrt{X}$
1
Ecr
0
X
c
2
1
0



Count
2018
19
25
2034
0
500
1000
1500
2000
00
01
10
11

IBM Quantum Platform

https://quantum.ibm.com/



Ⅲ.
Setting Up the Qiskit Development Environment

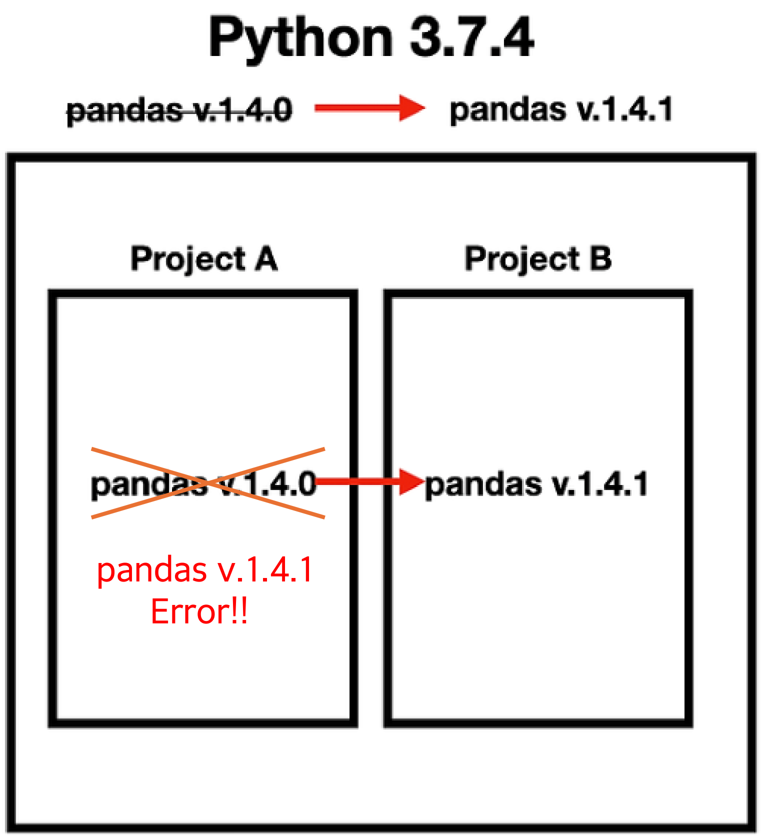
Virtual Environment

Virtual environments are used to create isolated workspaces.

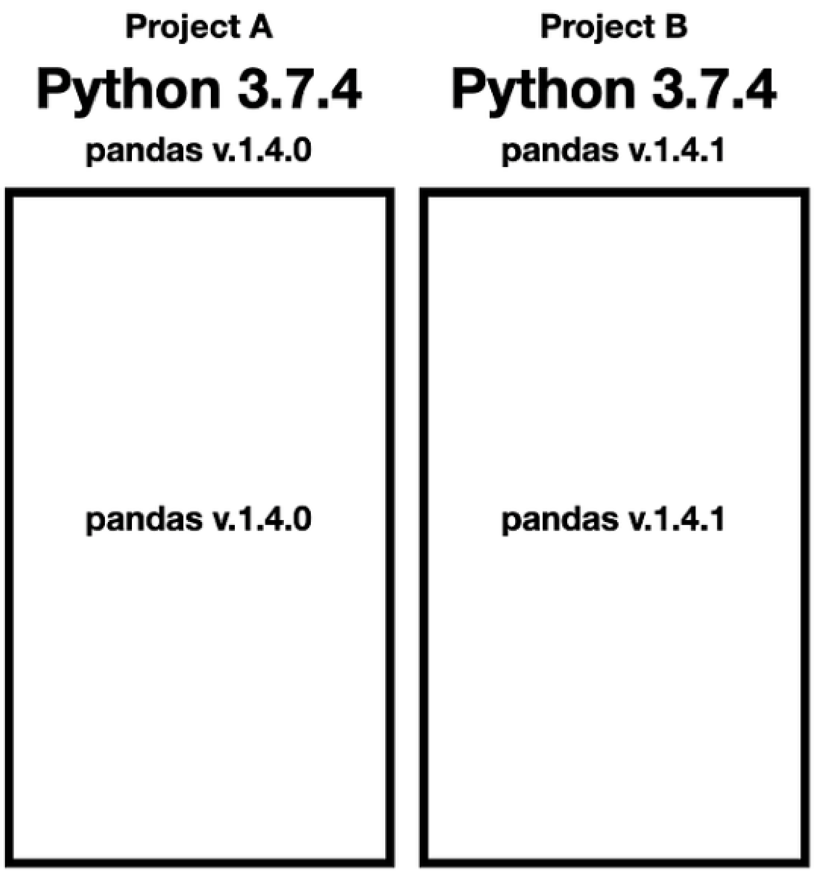
As the number of dependencies between libraries increases, conflicts between libraries can become more frequent, and in the worst case, you might have to delete everything and set up the development environment from scratch.

To avoid these issues, it's helpful to create a separate virtual environment for each project, where only the necessary libraries are installed.

Virtual Environment



<Virtual Environment X>



<Virtual Environment O>

image source: https://heytech.tistory.com/316

Setting Up the Environment

Contents of “Setting Qiskit Development Environment.docx”

1. Why Virtual Environments Are Necessary
2. Installing Anaconda
3. Getting Familiar with Anaconda Prompt
4. Creating a Virtual Environment
5. Activating the Virtual Environment
6. Installing Libraries in the Virtual Environment
7. Running and Testing Jupyter Notebook

IV.
Designing Quantum Circuits

QuantumCircuit Class

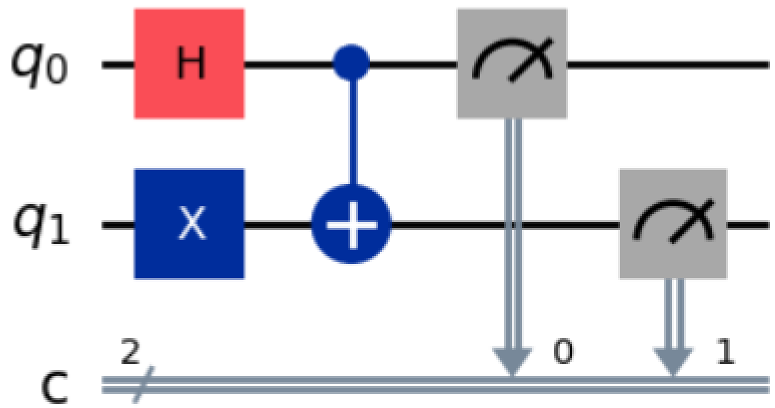
To build a circuit:

- Initialize a quantum circuit object
- Perform operations on those qubits

[4]:

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2,2)

qc.h(0)
qc.x(1)
qc.cx(0, 1)

qc.measure([0,1], [0,1])

qc.draw("mpl")
```

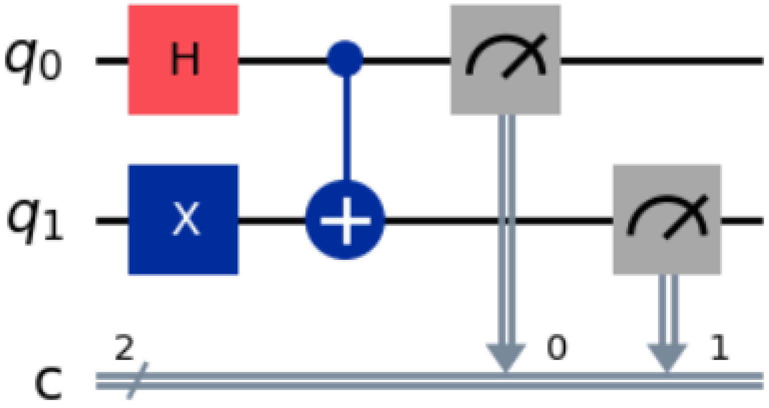
[4]:



QuantumCircuit Class

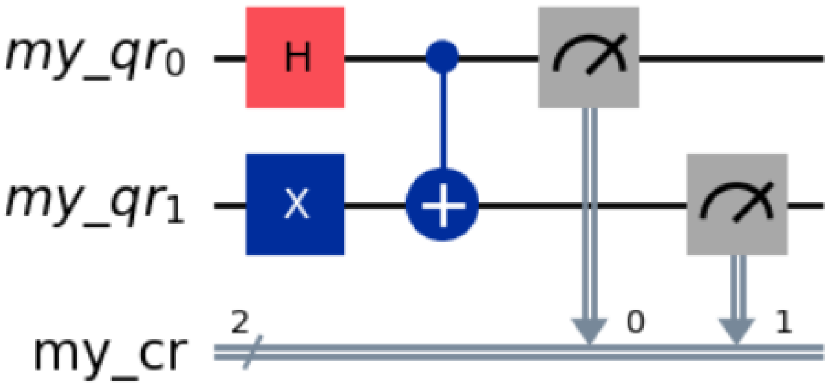
[4]:
```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(2,2)

qc.h(0)
qc.x(1)
qc.cx(0, 1)

qc.measure([0,1], [0,1])

qc.draw("mpl")
```

[4]:



[5]:
```python
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister

qr = QuantumRegister(2, name="my_qr")
cr = ClassicalRegister(2, name="my_cr")
qc = QuantumCircuit(qr,cr)

qc.h(0)
qc.x(1)
qc.cx(0, 1)

qc.measure([0,1], [0,1])

qc.draw("mpl")
```

[5]:



Simulator and Real Backend

Simulators

- Software-based backend that mimic the behavior of a quantum computer
- They are usually used for testing and experimenting with quantum algorithms without relying on real quantum hardware.
- As the number of qubits increases, the classical resources required to simulate them grow exponentially, making simulators impractical for large-scale systems

Real backend

- Actual quantum processors accessible via the cloud.
- They allow users to run quantum circuits on physical quantum hardware.
- Subject to real-world constraints, such as noise, decoherence, and gate errors.
- Limited by hardware-specific characteristics, including qubit connectivity (topology) and maximum number of qubits.

Simulator and Real Backend

Simulators Example: qiskit_aer.AerSimulator

- Software-based backend that mimic the behavior of a quantum computer
- They are usually used for testing and experimenting with quantum algorithms without relying on real quantum hardware.
- As the number of qubits increases, the classical resources required to simulate them grow exponentially, making simulators impractical for large-scale systems

Real backend Example: ibm_sherbrooke

- Actual quantum processors accessible via the cloud.
- They allow users to run quantum circuits on physical quantum hardware.
- Subject to real-world constraints, such as noise, decoherence, and gate errors.
- Limited by hardware-specific characteristics, including qubit connectivity (topology) and maximum number of qubits.

V.
Implementing Classical Feedforward and Control Flow

Classical Feedforward and Control Flow

- Classical feedforward and control flow are essential concepts in quantum computing that involve using classical measurement results to dynamically control quantum operations.
- Many quantum algorithms, such as the HHL algorithm, require post-measurement processing to determine subsequent actions

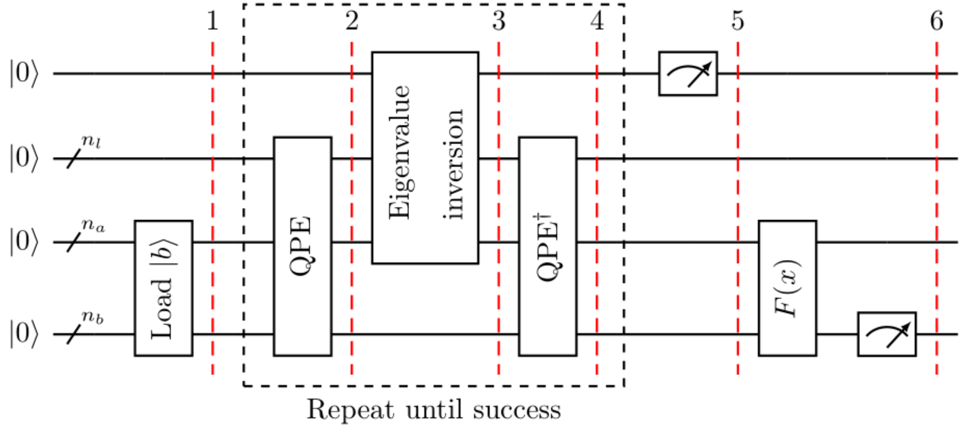


image source: qiskit-textbook

Types of Statements for Classical Feedforward

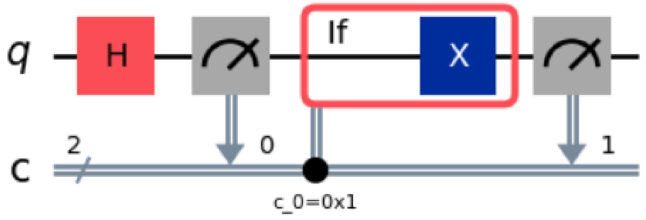
- If statement
- Switch statement
- For loop
- While loop
- Break loop & Continue loop

If statement

[3]:

```python
qr = QuantumRegister(1, name='q')
cr = ClassicalRegister(2, name='c')
qc = QuantumCircuit(qr, cr)

# unpack the qubit and classical bits from the registers
(q0,) = qr
b0, b1 = cr

# apply Hadamard
qc.h(q0)
# measure
qc.measure(q0, b0)

# begin if test block. the contents of the block are executed if b0 == 1
with qc.if_test((b0, 1)):
    # if the condition is satisfied (b0 == 1), then flip the bit back to 0
    qc.x(q0)
# finally, measure q0 again
qc.measure(q0, b1)

qc.draw(output="mpl", idle_wires=False)
```

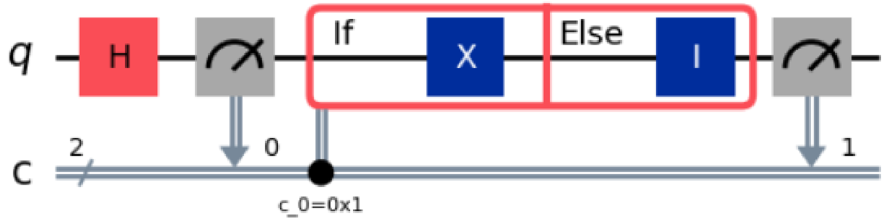
[3]:



[6]:

```python
from qiskit import QuantumCircuit
from qiskit.circuit import QuantumRegister, ClassicalRegister

qr = QuantumRegister(1, name='q')
cr = ClassicalRegister(2, name='c')
qc = QuantumCircuit(qr, cr)

# unpack the qubit and classical bits from the registers
(q0,) = qr
b0, b1 = cr

# apply Hadamard
qc.h(q0)
# measure
qc.measure(q0, b0)

# begin if test block. the contents of the block are executed if b0 == 1
with qc.if_test((b0, 1)) as else_:
    # if the condition is satisfied (b0 == 1), then flip the bit back to 0
    qc.x(q0)


with else_:
    # if the condition is satisfied (b0 != 1), then apply identity operator to 0
    qc.id(q0)
# finally, measure q0 again
qc.measure(q0, b1)

qc.draw(output="mpl", idle_wires=False)
```

[6]:

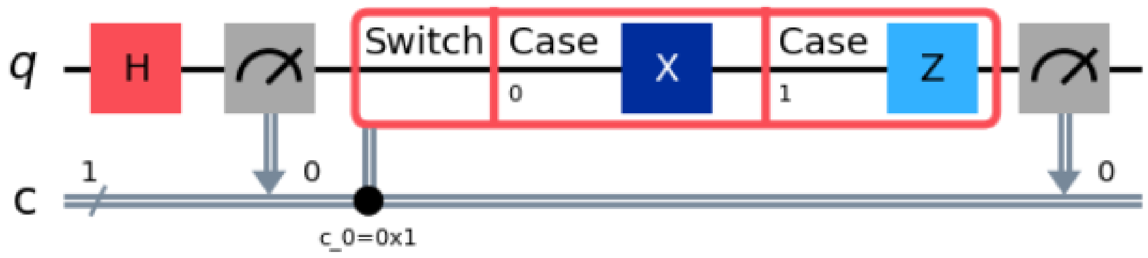


Switch statement

[12]:

```
qubits = QuantumRegister(1, name='q')
clbits = ClassicalRegister(1, name='c')
circuit = QuantumCircuit(qubits, clbits)
(q0,) = qubits
(c0,) = clbits

circuit.h(q0)
circuit.measure(q0, c0)
with circuit.switch(c0) as case:
    with case(0):
        circuit.x(q0)
    with case(1):
        circuit.z(q0)
circuit.measure(q0, c0)

circuit.draw("mpl")

# example output counts: {'1': 1024}
```

[12]:

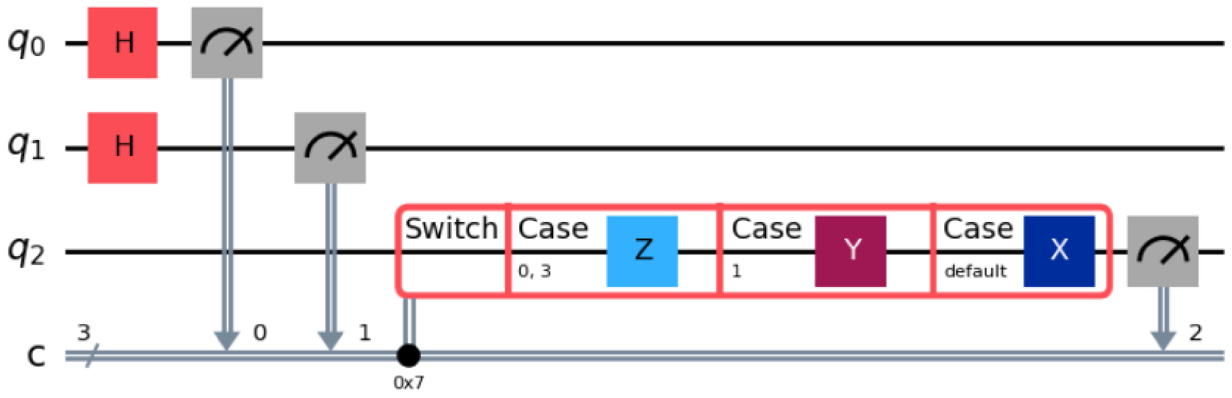


Switch statement

•[14]:

```
qubits = QuantumRegister(3, name='q')
clbits = ClassicalRegister(3, name='c')
circuit = QuantumCircuit(qubits, clbits)
(q0, q1, q2) = qubits
(c0, c1, c2) = clbits

circuit.h([q0, q1])
circuit.measure(q0, c0)
circuit.measure(q1, c1)
with circuit.switch(clbits) as case:
    with case(0b000, 0b011):
        circuit.z(q2)
    with case(0b001):
        circuit.y(q2)
    with case(case.DEFAULT):
        circuit.x(q2)
circuit.measure(q2, c2)

circuit.draw("mpl")
```

[14]:

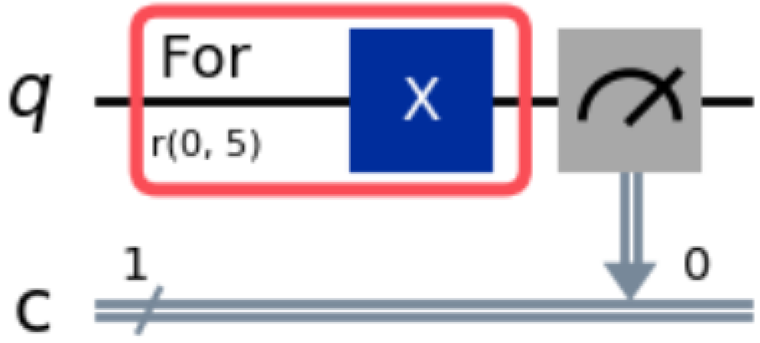


For loop

[15]:

```
qubits = QuantumRegister(1, name='q')
clbits = ClassicalRegister(1, name='c')
circuit = QuantumCircuit(qubits, clbits)
(q0,) = qubits
(c0,) = clbits

with circuit.for_loop(range(5)) as _:
    circuit.x(q0)
circuit.measure(q0, c0)

circuit.draw("mpl")
```

[15]:

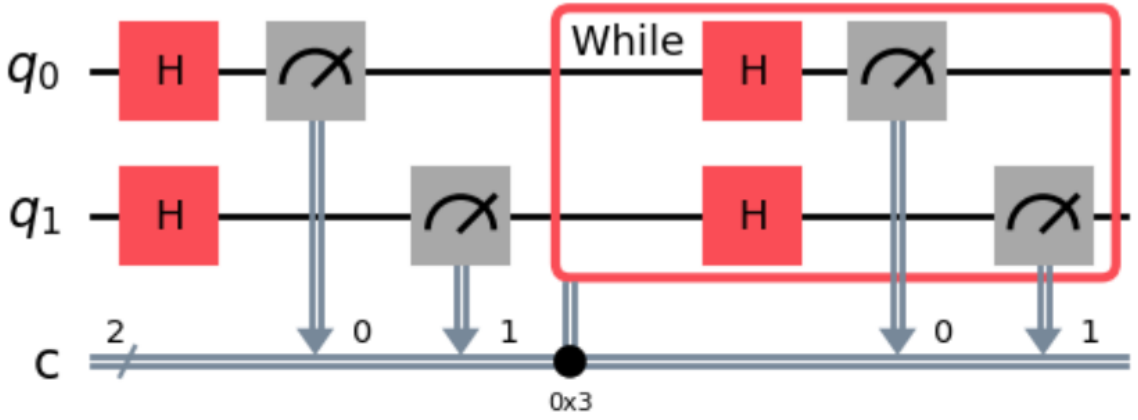


While loop

[16]:

```
qubits = QuantumRegister(2, name='q')
clbits = ClassicalRegister(2, name='c')
circuit = QuantumCircuit(qubits, clbits)

q0, q1 = qubits
c0, c1 = clbits

circuit.h([q0, q1])
circuit.measure(q0, c0)
circuit.measure(q1, c1)
with circuit.while_loop((clbits, 0b11)):
    circuit.h([q0, q1])
    circuit.measure(q0, c0)
    circuit.measure(q1, c1)

circuit.draw("mpl")
```

[16]:



Break loop

[18]:

```python
# Prepare quantum and classical bits
qubits = QuantumRegister(1, name='q')
clbits = ClassicalRegister(1, name='c')
circuit = QuantumCircuit(qubits, clbits)
(q0,) = qubits
(c0,) = clbits

# A loop to flip the qubit 10 times
with circuit.for_loop(range(10)) as i:
    circuit.x(q0)                   # Apply X gate to flip the qubit state
    circuit.measure(q0, c0)         # Measure the state of q0
    with circuit.if_test((clbits, 1)):
        circuit.break_loop()  # Break the loop if the measurement result is 1

# Final measurement
circuit.measure(q0, c0)

# Visualize the circuit
circuit.draw('mpl')
```

[18]: