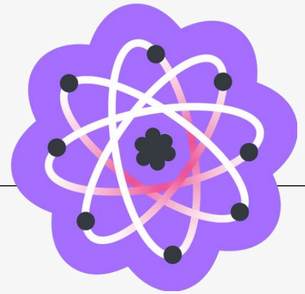
Qiskit Fall Fest 2025 @ Yonsei

Getting Started With Qiskit



Nov. 1, 2025

김정환
IT융합공학과 22학번 j.hwankim@yonsei.ac.kr

QIYA

CONTENTS

QISKIT 2025 2025 FALL FEST

QIYA

01. Qiskit이란?

IBM에서 개발한 양자 회로를 설계하고 클라우드를 통해 양자 컴퓨터에 접근하거나 시뮬레이션하기 위해 만들어진 Python 기반 오픈소스 퀀텀 컴퓨팅 SDK

Qiskit

QISKIT 2025 2025 FALL FEST

QIYA

01. Qiskit이란?

Qiskit을 사용하는 이유

- Python 기반: 데이터 과학, AI 분야에서 가장 널리 쓰이는 파이썬을 사용합니다.
- 오픈소스: 누구나 무료로 사용하고, 기여하고, 코드를 볼 수 있습니다.
- 클라우드를 통해 실제 IBM 양자 컴퓨터에 내가 만든 회로를 전송하고 실행할 수 있습니다.

IonQ, Amazon Bracket(Rigetti, OQC 등) 등 IBM Cloud외의 다른 클라우드 서비스도 Qiskit으로 사용가능합니다.

- 강력한 로컬/클라우드 시뮬레이터도 제공합니다.
- 활발한 커뮤니티와 방대한 학습 자료(튜토리얼 등)들이 있습니다.

QIYA

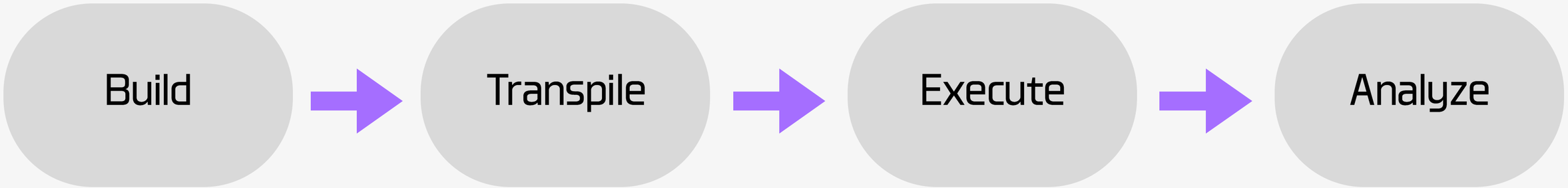
01. Qiskit이란?

다른 양자컴퓨터 프로그래밍 언어

- Pennylane: Xanadu에서 개발한 QML, Differential Programming에 강점이 있는 Python 라이브러리. PyTorch, TensorFlow 같은 머신러닝 도구와 자연스럽게 결합됩니다.
- Circq: Google에서 개발한 NISQ 알고리즘 연구에 중점을 둔 Python 라이브러리.
- Q#: Microsoft에서 개발한 언어. C#과 문법이 비슷합니다. Quatum Azure 클라우드와 연동됩니다.

QIYA

02. Qiskit의 4단계

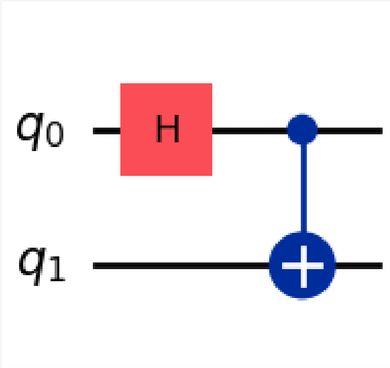


03. 회로 만들기

- Formulation : 풀고 싶은 문제를 양자 알고리즘 (양자 회로)으로 설계합니다.
- Implementation (코드 구현): 설계한 알고리즘을 Qiskit 코드로 옮깁니다.

ex) Bell State 만들기:

- 2 큐비트 회로 생성
- 0번 큐비트에 H게이트
- 0번과 1번 큐비트에 CX 게이트




```python
from qiskit import QuantumCircuit

# 2개의 큐비트가 있는 회로 생성
qc = QuantumCircuit(2)

# 0번 큐비트에 H게이트 (중첩)
qc.h(0)
# 0번과 1번 큐비트에 CNOT게이트 (얽힘)
qc.cx(0, 1)

qc.draw('mpl') # 회로도 그리기
```

(참고) Qiskit의 큐비트 순서

- Big Endian vs Little Endian

컴퓨터가 bit 순서를 읽는 방식에는 크게 두 가지가 있습니다.
Big endian은 그냥 앞에서부터 읽는 방식이고, little endian은 뒷자리부터 읽는 방식입니다.
ex) 1011 → 1011(BE), 1101(LE)

- Qiskit은 bitstring을 little-endian 표기법으로 표기합니다.

ex) 결과 bitstring이 11000로 나왔다면, [q0=0, q1=0, q2=0, q3=1, q4=1]입니다.
그냥 Qiskit의 bitstring은 거꾸로 해석하시면 됩니다.

QIYA

03. 회로 만들기

QASM (Quantum Assembly Language)

QASM은 양자 회로를 텍스트로 표현하는 표준 언어입니다.
우리가 만든 Qiskit의 QuantumCircuit 객체를 실제 하드웨어나
시뮬레이터가 이해할 수 있는 어셈블리어 입니다.

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
h q[0];
cx q[0],q[1];
```

QIYA

04. Transpiling

Transpiling 이란?

Build 단계에서 만든 이상적인 회로를 실제 양자 하드웨어가 실행할 수 있는 형태로 translate하고 최적화하는 과정입니다. 고전컴퓨터에서의 컴파일링에 대응되는 단계입니다.

QIYA

04. Transpiling

Transpiling이 필요한 이유 1)

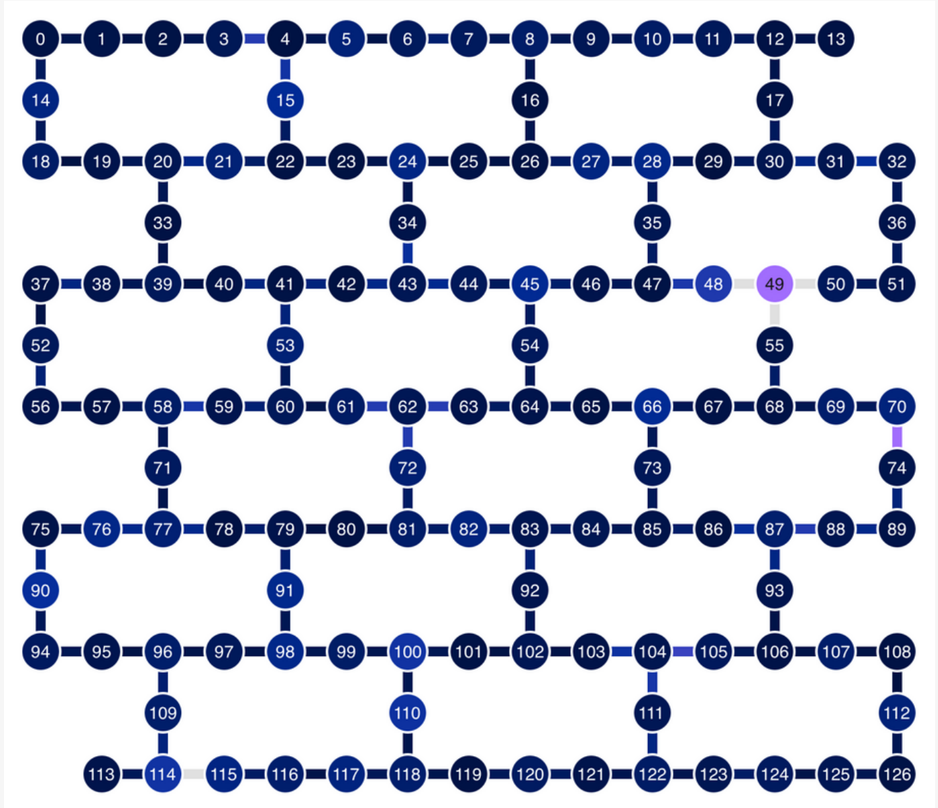
실제 양자 칩에서는 모든 큐비트가 서로 직접 연결되어 있지 않기 때문에 transpiling을 통해 실제 QPU에서 실행 가능하도록 변형해줘야합니다.

QIYA

04. Transpiling

ex) 칩이 실제로는 (0-1), (1-2)번 큐비트만 연결돼있는데, qc.cx(0,2)가 회로에 있다면, SWAP 게이트를 써서 (0, 2) CNOT: (1, 2) SWAP → (0, 1) CNOT → (1, 2) SWAP 과 같이 명령을 바꿉니다.

그 결과 회로가 더 복잡해지고 커집니다.



IBM Eagle 프로세서 큐비트 맵

04. Transpiling

- Transpiling이 필요한 이유 2)
    - 실제 양자 칩에는 우리가 사용하는 모든 게이트(H, X, CNOT, ...)들이 다 구현되어있지 않습니다.
    - 네이티브 게이트 (Native Gates): 각 하드웨어가 실제로 실행할 수 있는 기본 부품 같은 게이트들만 정해져 있습니다. (예: ECR, SX, RZ 게이트)
    - 우리가 쓴 이상적인 게이트를 네이티브 게이트의 조합으로 분해합니다.
      ex)H → RZ(π/2)·SX·RZ(π/2), CNOT → SX·RZ(π)·...·ECR·...

QISKIT 2025 2025 FALL FEST

QIYA

04. Transpiling

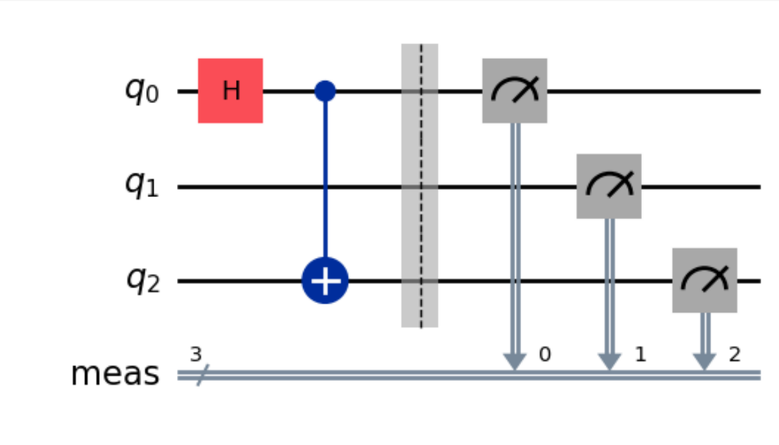
- Transpiling이 필요한 이유 3)

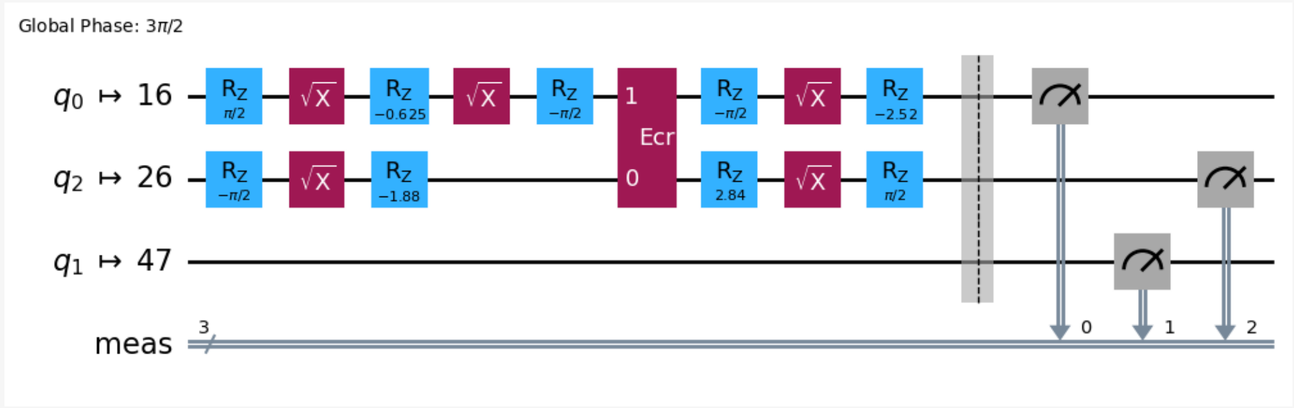
    - Transpiler는 단순히 회로를 번역만 하는 것이 아니라, 최대한 짧고 효율적으로 만듭니다.

      ex) qc.x(0) → qc.x(0) (X 게이트를 연속 두 번 = 아무것도 안 함)

      Transpiler가 이런 부분을 알아서 제거해줍니다.

04. Transpiling





QIYA

05. Execution

Transpile된 회로를 Backend에서 실제로 구동시키고, 그 결과를 Job으로 돌려받는 단계입니다.

QIYA

05. Execution

- Backend 종류: Local Simulator
    - 실제 양자 하드웨어를 수학적으로 흉내 내는 가상 백엔드입니다. (주로 로컬에서 실행)
    - Qiskit 2.x 에서는 AerSimulator로 통합되었습니다.
    - Noiseless simulation을 하거나, noise를 추가하거나, 실제 하드웨어의 noise를 가져와서 noisy simulation이 가능합니다.

05. Execution

- Backend 종류: Local Simulator
    - 시뮬레이션을 할때 필요한 메모리는 큐비트 수에 따라 지수적으로 증가합니다.
      일반 노트북 (16GB~32GB): 최대 28~31 큐비트
      워크스테이션 (256GB): 최대 33~34 큐비트
      슈퍼컴퓨터 (페타바이트급): 최대 40~50 큐비트
    - 메모리에 저장된 거대한 상태 벡터에 게이트 연산(행렬 곱셈)을 적용하고, shots 수만큼 샘플링을 해야합니다.
    - AerSimulator는 GPU(NVIDIA CUDA)를 사용한 병렬 처리를 지원합니다.

05. Execution

- Backend 종류: Cloud
  - 클라우드를 통해 실제 양자 컴퓨터를 사용할 수 있습니다.
  - 실제 하드웨어이기 때문에, 시뮬레이션과 달리 노이즈가 있습니다.
  - Qiskit은 provider를 통해 다양한 클라우드를 사용할 수 있습니다.

    qiskit-ibm-provider: IBM Quantum의 초전도 하드웨어

    qiskit-ionq: IonQ의 이온 트랩 하드웨어

    qiskit-braket-provider (AWS), qiskit-quantinuum-provider (Quantinuum) 등

05. Execution

- Backend 종류: Cloud
    - 클라우드 하드웨어는 여러 사용자가 공유하기 때문에 대기열이 있습니다.
    - 사용하고자 하는 프로바이더의 계정과 API Key가 있어야합니다.
    - 일부 하드웨어는 짧은 시간동안 무료로 개방되어있지만, 고성능 하드웨어나 장시간 사용하면 유료 결제를 해야합니다.

QIYA

05. Execution

- Qiskit Primitives
    - 양자 하드웨어 또는 시뮬레이터를 단순하고 효율적으로 활용하기 위한 표준화된 인터페이스.
    - Qiskit에서 회로를 “실행”하거나 “측정”하는 과정을 더 추상화한 high level 실행 계층이에요.
    - 이 계층은 모든 백엔드(AerSimulator, IBM Quantum 등)에서 공통으로 동작하도록 설계되었습니다.

QIYA

05. Execution

- Qiskit Primitives: Sampler
    - 회로를 실행하고 측정 확률 분포(probability distribution)를 반환
    - 회로를 shots 수만큼 실행(샘플링)하고, 각 결과 비트스트링(00, 01...)이 나올 확률을 반환합니다.

QIYA

05. Execution

- Qiskit Primitives: Estimator
    - 회로와 Observable을 입력으로 받아 기대값($\langle\psi|O|\psi\rangle$)을 계산합니다.
    - VQE, QAOA 같은 하이브리드 알고리즘(VQA)에 필수적입니다.

05. Execution

- Execution Modes
    - Job Mode: sampler.run()을 호출할 때마다 새로운 workload를 1회성으로 하드웨어에 제출합니다.
    - Session Mode: 하드웨어와 연결을 유지하는 세션을 열고, 그 안에서 여러 Job을 연속으로 주고받아 다시 대기열을 기다릴 필요가 없습니다.

06. Analyze

- Execute 단계에서 반환된 result 객체에서 데이터를 추출하여, 사람이 해석 가능한 정보(그래프, 값)로 후처리하는 최종 단계입니다.

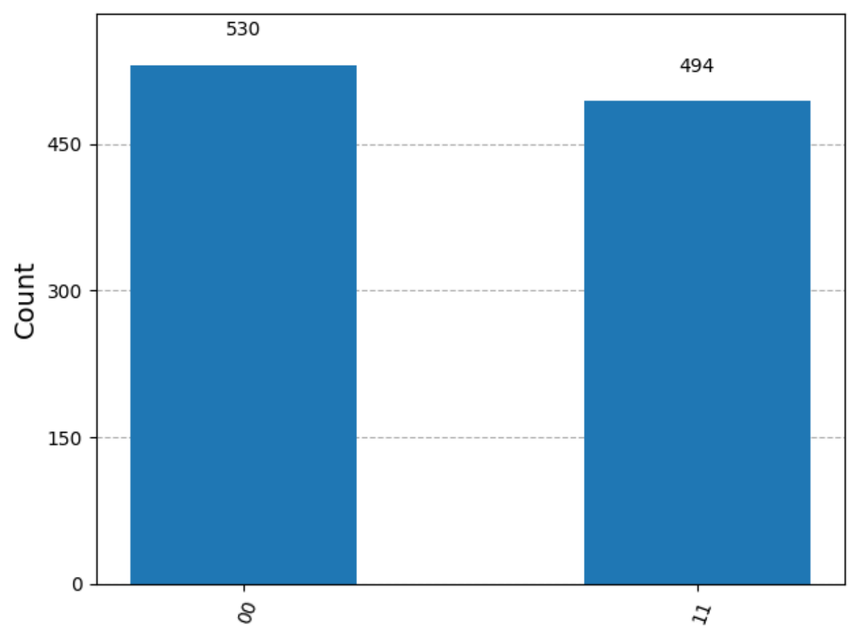
  ex) Sampler로 얻은 확률 분포를 히스토그램으로 그리기

  Estimator로 얻은 기댓값을 VQE 같은 알고리즘의 최적화 루프의 다음 입력값으로 사용하기

- Result 객체에는 실행 결과 뿐 아니라 다양한 메타데이터들이 있습니다.

QIYA

06. Analyze



<Sampler 결과>

```
job = estimator.run([(qc, observable)])
exp_value = job.result()[0].data.evs.item()

print(f"Bell State 기댓값: {exp_value}")
```

✓ 0.0s

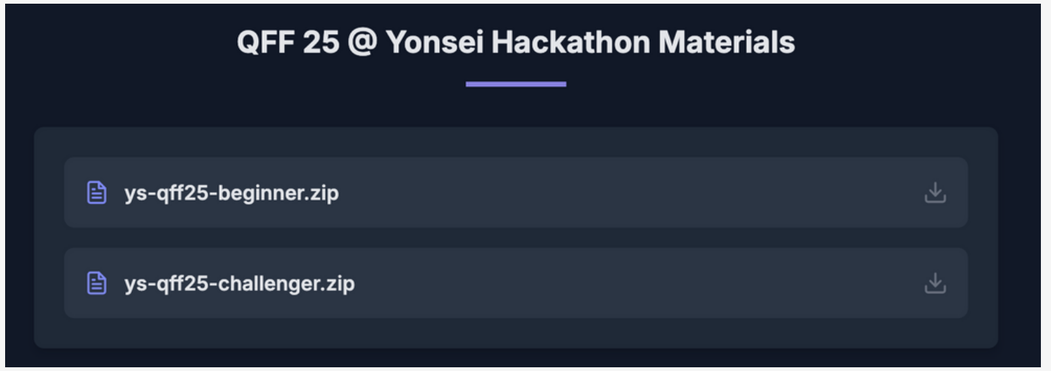
```
Bell State 기댓값: 1.0
```

<Estimator 결과>

QIYA

07. Hackathon 소개

https://qiya-yonsei.github.io



- Beginner: Qiskit을 처음 사용하시는 분들을 위해 Qiskit 기본 문법부터 다룹니다.
- Challenger: Qiskit에 익숙한 분들을 위한 응용 주제 문제들이 있습니다.

자유롭게 도전해보고 싶은 난이도 도전하세요!
둘 다 하셔도 됩니다

07. Hackathon 소개

- 해커톤 경품

Beginner 경품

완료자 중 추첨을 통해 3명에게
45W 20,000mAh 보조 배터리 증정

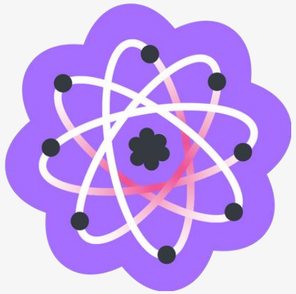

Challenger 경품

완료자 중 챌린지 점수 상위권자에게 경품 증정

AirPods Pro (1명)
BOSE Soundlink Flex (2명)
45W 20,000mAh 보조배터리 (3명)

QIYA

THANK YOU



QIYA