

# SENG 457/CSC 557

## Lab 2: Qiskit and PennyLane

---

Prashanti Priya Angara, Maziyar Khadivi

Contact email: [mazy1996@uvic.ca](mailto:mazy1996@uvic.ca)

May 20, 2025

## To-do list for today

- Get started with Qiskit
- Get started with PennyLane
- Sign the attendance sheet!

## Activity 1: Hello Circuit

- Reduce the number of quantum registers to 1
- Reduce the number of classical registers to 1

Note that we can manipulate the number of registers by also going to Edit > Manage Registers

- What do we see under:
  - Probabilities: ?
  - Q-Sphere: ?
  - Statevector: ?

## Activity 2: Flip the state

- Transform the state of  $q[0]$  from  $|0\rangle$  to  $|1\rangle$
- What gate(s) would we apply?
- What do we see under:
  - Probabilities: ?
  - Q-Sphere: ?
  - Statevector: ?

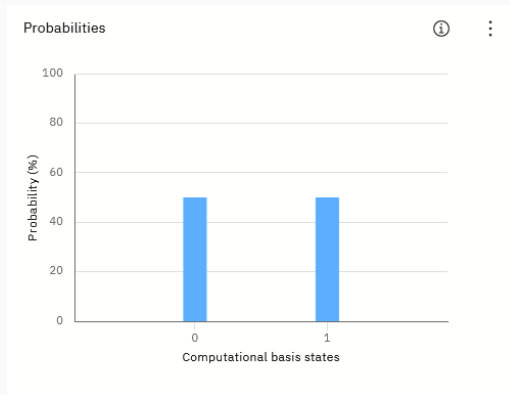
## Activity 3: Superposition and Grouping

- Transform the state of  $q[0]$  to the equal superposition state  $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$
- Group the gates involved and call it the “MinusOp”
- What gate(s) would we apply?
- What do we see under:
  - Probabilities: ?
  - Q-Sphere: ?
  - Statevector: ?

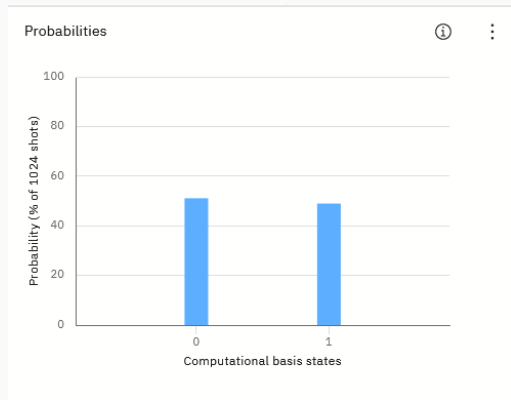
## Activity 4: Measurement

- Append a measurement operation to `q[0]` which was in the same state as in Activity 3, i.e.,  $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$
- What do we see under:
  - Probabilities: ?
  - Q-Sphere: ?
  - Statevector: ?

# A note on probabilities and measurement



(a) Without measurement (analytical calculation)



(b) With measurement (out of 1024 shots)

**Figure 1:** In Figure 1a, probabilities shown are  $|\alpha|^2$  and  $|\beta|^2$ . Figure 1b shows results when the circuit is run 1024 times, which is uneven due to “shot noise”.

# Features: Running on actual quantum machines

- Click on “Setup and Run” on the right top corner in the composer to see available machines to run your quantum experiments on.

The screenshot displays the IBM Quantum Learning Composer interface. On the left, the 'Untitled circuit' workspace shows a quantum circuit with two qubits,  $q[0]$  and  $c01$ . The circuit includes a Hadamard gate on  $q[0]$ , followed by a CNOT gate with  $q[0]$  as control and  $c01$  as target, and another Hadamard gate on  $q[0]$ . Below the circuit, the 'Probabilities' section shows a bar chart for computational basis states, with state 1 having a probability of 100%. The 'Q-sphere' visualization is also visible.

On the right, the 'Set up and run your circuit' dialog is open, showing two steps:

**Step 1: Choose a system or simulator**

Search by system or simulator name

Available systems/simulators:

- ibm\_kyoto** (Selected)
  - System status: Online
  - Total pending jobs: 32
  - 127 Qubits, 3.6% EPLG, 5K CLOPS
  - [See details](#)
- simulator\_stabilizer**
  - Simulator status: Online
  - Total pending jobs: 0
  - 5000 Qubits, N/A EPLG, N/A CLOPS
  - [See details](#)
- simulator\_mps**
  - Simulator status: Online
  - Total pending jobs: 0
  - 100 Qubits, N/A EPLG, N/A CLOPS
  - [See details](#)
- simulator\_extended\_stabilizer**
  - [See details](#)

**Step 2: Choose your settings**

Instance: ibm-q/open/main

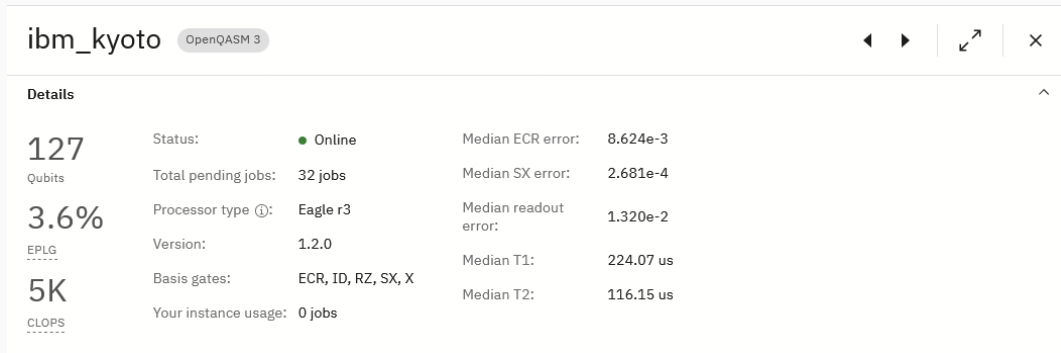
Shots: 1

Job limit: 3 remaining

Tags (optional): Add tags



- EPLG: Error Per Layered Gate
- CLOPS: Circuit Layer Operations Per Second



Two good articles: 1-How we measure quantum quality and speed  
2- Three Metrics for Quantum Computing Performance: Scale, Quality and Speed

# The Bloch Sphere

- The Bloch sphere is a geometric representation used in quantum mechanics to visualize the state of a single qubit, which is the basic unit of quantum information.
- Let's navigate to: <https://javafxpert.github.io/grok-bloch/>

- The  $q$ -sphere represents the state of a system of one or more qubits (the Bloch sphere can represent only one qubit)
- The quantum amplitude is given as  $\sqrt{p_k}e^{i\phi_k}$
- $\sqrt{p_k}$  is the magnitude of the amplitude
- $e^{i\phi_k}$  is the **phase** and is calculated as

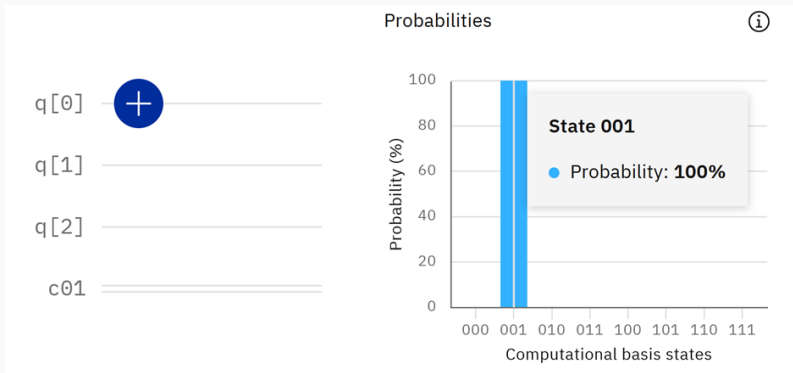
$$e^{i\phi_k} = \cos(\phi_k) + i \sin(\phi_k)$$

- For example, given a state  $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ , the magnitude of amplitude of the  $|1\rangle$  state is  $\frac{1}{\sqrt{2}}$  and the phase is calculated as  $e^{i\pi} = -1$

The four steps to writing a quantum program using Qiskit Patterns are:

- Map the problem to a quantum-native format.
- Optimize the circuits and operators.
- Run the AerSimulator for simulation or IBM Runtime for execution on QPU.
- Analyze the results.

## Qiskit Bit Ordering (Right to Left)

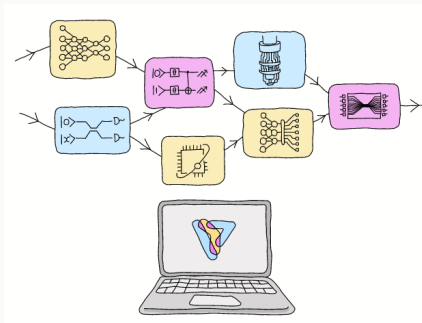


**PennyLane**

---

# QNode

- Represents a quantum node in the hybrid computational graph.
- A quantum node contains a quantum function (corresponding to a variational circuit) and the computational device it is executed on.
- The QNode calls the quantum function to construct a QuantumTape instance representing the quantum circuit.



This defines the device that the quantum circuit should be simulated/run on

- 'default.qubit': A simple state simulator of qubit-based quantum circuit architectures.
- 'default.mixed': A mixed-state simulator of qubit-based quantum circuit architectures.
- 'lightning.qubit': A more performant state simulator of qubit-based quantum circuit architectures written in C++.

## Device

Quantum circuit creation: The program creates a quantum circuit using the `qml.qnode` decorator and the `dev` device object.

```
dev = qml.device('default.qubit', wires=1, shots=1024)
```

<sup>1</sup><https://docs.pennylane.ai/en/stable/code/api/pennylane.device.html>



For any quantum node, the following values can be returned:

- Statevector
- Probabilities
- Expectation Values
- Samples

## Sample Program: Returning Probabilities

```
[16]: dev = qml.device('default.qubit', wires=1, shots=1024)
```

```
@qml.qnode(dev)
def circuit():
    qml.Hadamard(wires=0)
    return qml.probs(wires=0)
```

```
[17]: print("Probabilities:", circuit())
```

```
Probabilities: [0.47949219 0.52050781]
```

## Sample Program: Returning the Statevector

- Note that for returning a Statevector, you should not pass the shots parameter in the device

```
[13]: dev = qml.device('default.qubit', wires=1)
```

```
@qml.qnode(dev)
def circuit():
    qml.Hadamard(wires=0)
    return qml.state()
```

```
[14]: print("State:", circuit())
```

```
State: [0.70710678+0.j 0.70710678+0.j]
```