

Spring 2024 3

Sam Bieberich



Announcements

- Thanks for coming back again
- QHack
 - <https://qhack.ai/online-events/#streaming-sessions>



What are we doing now?

We didn't finish the book last semester, but instead of just going chapter by chapter, we will be targeting the prerequisites for the exam:

<https://www.ibm.com/training/certification/ibm-certified-associate-developer-quantum-computation-using-qiskit-v02x-C0010300>

On next slide too...



Number of questions: 60

Number of questions to pass: 44

Time allowed: 90 minutes

Status: Live

| | | |
|--|-----|---|
| Section 1: Perform Operations on Quantum Circuits | 47% | ▼ |
| Section 2: Executing Experiments | 3% | ▼ |
| Section 3: Implement BasicAer: Python-based Simulators | 3% | ▼ |
| Section 4: Implement Qasm | 1% | ▼ |
| Section 5: Compare and Contrast Quantum Information | 10% | ▼ |
| Section 6: Return the Experiment Results | 7% | ▼ |
| Section 7: Use Qiskit Tools | 1% | ▼ |
| Section 8: Display and Use System Information | 3% | ▼ |
| Section 9: Construct Visualizations | 19% | ▼ |
| Section 10: Access Aer Provider | 6% | ▼ |



Section 1

Perform Operations on Quantum Circuits

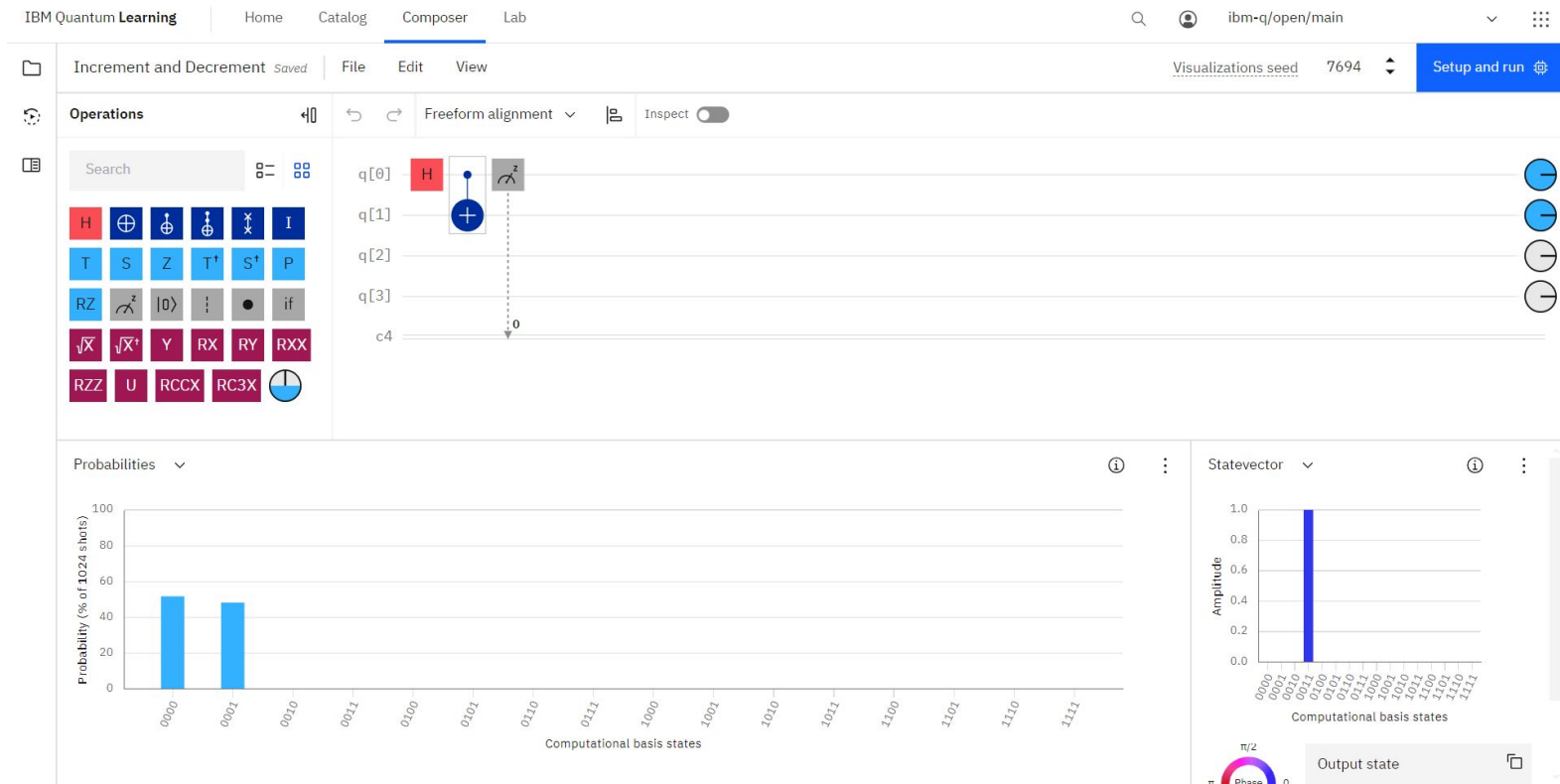


8 section topics

1. Construct multi-qubit quantum registers
2. Measure quantum circuits in classical registers
3. Use single-qubit gates
4. Use multi-qubit gates
5. Use barrier operations
6. Return the circuit depth
7. Extend quantum circuits
8. Return the OpenQASM string for a circuit



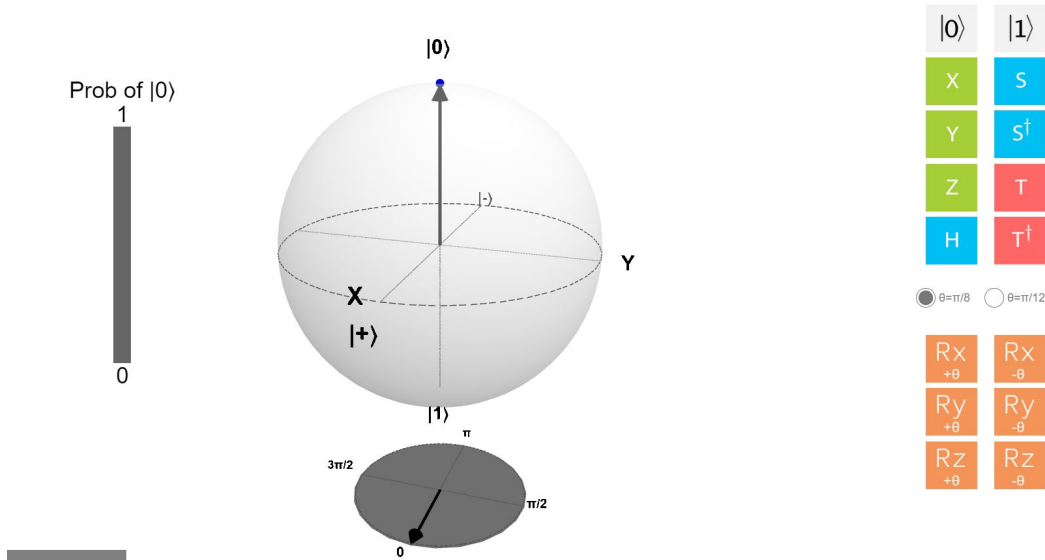
What's the composer?



Grok the Bloch Sphere

<https://javafxpert.github.io/grok-bloch/>

$$|\psi\rangle = \sqrt{1.00} |0\rangle + (\sqrt{0.00}) e^{i0} |1\rangle$$



Next Time

- Also it is recommended that we learn about all of the relevant examples in the IBMQ Lab
 - There are a lot of interesting tutorials on the quantum lab that we could look into in the coming weeks, as well as some free coursework in their catalog
 - More info on their website:
<https://learning.quantum.ibm.com/catalog/courses>



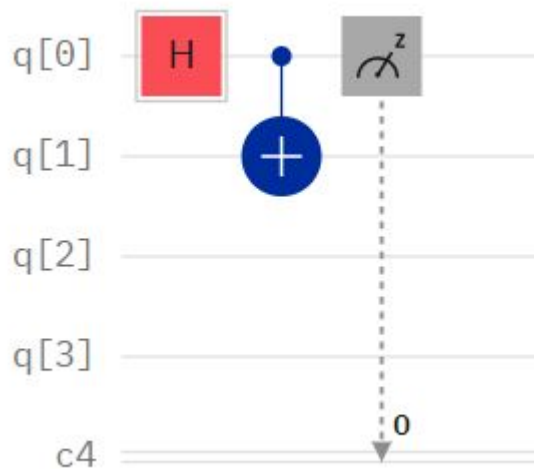
Construct multi-qubit quantum registers

Recap for how to make a quantum circuit with Qiskit:

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi
```

```
qreg_q = QuantumRegister(4, 'q')
creg_c = ClassicalRegister(4, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)
```

```
circuit.h(qreg_q[0])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.measure(qreg_q[0], creg_c[0])
```



Construct multi-qubit quantum registers

- For each quantum register, you need/should use a classical register that is the size of how many qubits you are reading:

Ex: 6 qubits, 2 ancilla, then classical register width = 4

- Often named **qreg** and **creg** so that they can be indexed easier



Construct multi-qubit quantum registers

Working through sample question #1

Familiarity with Qiskit API



1. Which statement will create a quantum circuit with four quantum bits and four classical bits?

- ✓ A. `QuantumCircuit(4, 4)`
- B. `QuantumCircuit(4)`
- C. `QuantumCircuit(QuantumRegister(4, 'qr0'), QuantumRegister(4, 'cr1'))`
- D. `QuantumCircuit([4, 4])`

qiskit.circuit.QuantumCircuit

`CLASS QuantumCircuit(*regs, name=None, global_phase=0, metadata=None)` [SOURCE]

Create a new circuit.

A circuit is a list of instructions bound to some registers.

Parameters

- **regs** (`list(Register)` or `list(int)` or `list(list(bit))`) –

The registers to be included in the circuit.

- If a list of `Register` objects, represents the `QuantumRegister` and/or `ClassicalRegister` objects to include in the circuit.

For example:

- `QuantumCircuit(QuantumRegister(4))`
- `QuantumCircuit(QuantumRegister(4), ClassicalRegister(3))`
- `QuantumCircuit(QuantumRegister(4, 'qr0'), QuantumRegister(2, 'qr1'))`

- If a list of `int`, the amount of qubits and/or classical bits to include in the circuit. It can either be a single int for just the number of quantum bits, or 2 ints for the number of quantum bits and classical bits, respectively.

For example:

- `QuantumCircuit(4)` # A QuantumCircuit with 4 qubits
- `QuantumCircuit(4, 3)` # A QuantumCircuit with 4 qubits and 3 classical bits

<https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html>

IBM Quantum



Construct multi-qubit quantum registers

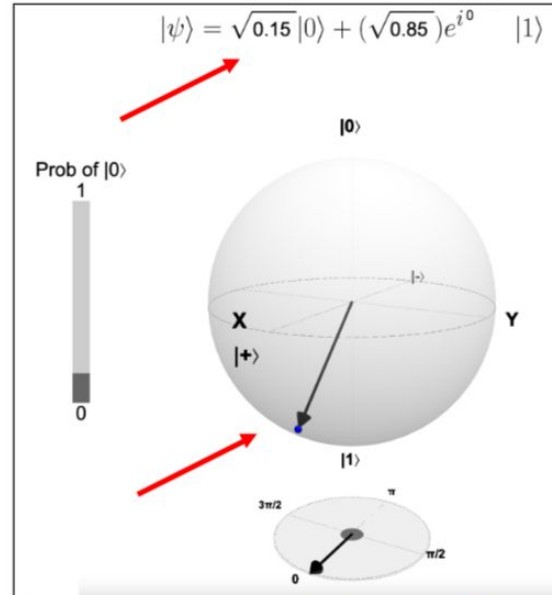
Working through sample question #2

Mental gymnastics on the Bloch sphere

2. Given this code fragment, what is the probability that a measurement would result in $|0\rangle$?

```
qc = QuantumCircuit(1)
qc.ry(3 * math.pi/4, 0)
```

- A. 0.8536
- B. 0.5
- ✓ C. 0.1464
- D. 1.0



<https://javafxpert.github.io/grok-bloch/>



IBM Quantum



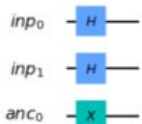
Working through sample question #3

Familiarity with Qiskit API

3. Assuming the fragment below, which three code fragments would produce the circuit illustrated?

```
inp_reg = QuantumRegister(2, name='inp')
ancilla = QuantumRegister(1, name='anc')
qc = QuantumCircuit(inp_reg, ancilla)
```

Insert code here



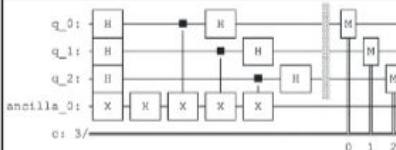
- ✓ A. `qc.h(inp_reg)`
`qc.x(ancilla)`
`qc.draw()`
- ✓ B. `qc.h(inp_reg[0:2])`
`qc.x(ancilla[0])`
`qc.draw()`
- ✓ C. `qc.h(inp_reg[0:1])`
`qc.x(ancilla[0])`
`qc.draw()`
- ✓ D. `qc.h(inp_reg[0])`
`qc.h(inp_reg[1])`
`qc.x(ancilla[0])`
`qc.draw()`
- E. `qc.h(inp_reg[1])`
`qc.h(inp_reg[2])`
`qc.x(ancilla[1])`
`qc.draw()`
- F. `qc.h(inp_reg)`
`qc.h(inp_reg)`
`qc.x(ancilla)`
`qc.draw()`

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

qr = QuantumRegister(3, 'q')
anc = QuantumRegister(1, 'ancilla')
cr = ClassicalRegister(3, 'c')
qc = QuantumCircuit(qr, anc, cr)
```

```
qc.x(anc[0])
qc.h(anc[0])
qc.h(qr[0:3])
qc.cx(qr[0:3], anc[0])
qc.h(qr[0:3])
qc.barrier(qr)
qc.measure(qr, cr)

qc.draw()
```



<https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html>

qiskit.circuit.QuantumRegister

CLASS `QuantumRegister(size=None, name=None, bits=None)` [SOURCE]

Implement a quantum register.

Create a new generic register.

Either the `size` or the `bits` argument must be provided. If `size` is not None, the register will be pre-populated with bits of the correct type.

Parameters

- `size` (*int*) – Optional. The number of bits to include in the register.
- `name` (*str*) – Optional. The name of the register. If not provided, a unique name will be auto-generated from the register type.
- `bits` (*list[Bit]*) – Optional. A list of `Bit()` instances to be used to populate the register.

<https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumRegister.html>

IBM
Associate
Certified

Developer
Quantum Computation
using Qiskit v0.2X

IBM Quantum

Another problem explained

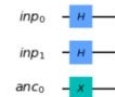
What has already been done?

1. Make a register called `inp_reg`
2. Make another register called `ancilla`
 - a. Both are quantum
3. Make a quantum circuit composed of these two quantum registers

3. Assuming the fragment below, which three code fragments would produce the circuit illustrated?

```
inp_reg = QuantumRegister(2, name='inp')
ancilla = QuantumRegister(1, name='anc')
qc = QuantumCircuit(inp_reg, ancilla)
```

Insert code here



- ✓ A. `qc.h(inp_reg)`
`qc.x(ancilla)`
`qc.draw()`
- ✓ B. `qc.h(inp_reg[0:2])`
`qc.x(ancilla[0])`
`qc.draw()`
- C. `qc.h(inp_reg[0:1])`
`qc.x(ancilla[0])`
`qc.draw()`
- ✓ D. `qc.h(inp_reg[0])`
`qc.h(inp_reg[1])`
`qc.x(ancilla[0])`
`qc.draw()`
- E. `qc.h(inp_reg[1])`
`qc.h(inp_reg[2])`
`qc.x(ancilla[1])`
`qc.draw()`
- F. `qc.h(inp_reg)`
`qc.h(inp_reg)`
`qc.x(ancilla)`
`qc.draw()`

Another problem explained

What do we need to do?

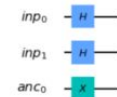
1. Add H gates to inp index 0 and 1
2. Add x to ancilla register index 0 (only index)
3. Draw the circuit (qc.draw())

Note the difference between B and C, the bounds [0:2] are non-inclusive for the upper bound so it is (0,1) indexes.

3. Assuming the fragment below, which three code fragments would produce the circuit illustrated?

```
inp_reg = QuantumRegister(2, name='inp')
ancilla = QuantumRegister(1, name='anc')
qc = QuantumCircuit(inp_reg, ancilla)
```

Insert code here

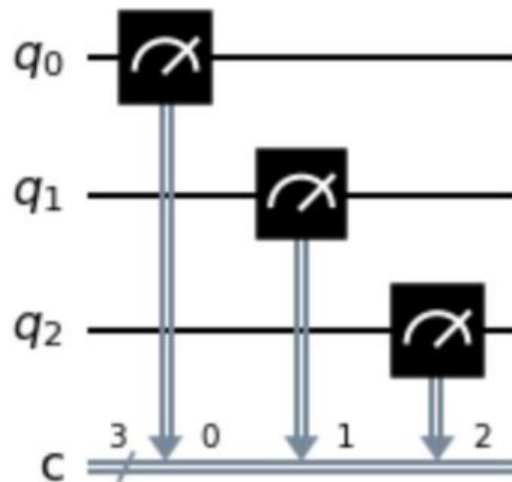


- ✓ A. `qc.h(inp_reg)`
`qc.x(ancilla)`
`qc.draw()`
- ✓ B. `qc.h(inp_reg[0:2])`
`qc.x(ancilla[0])`
`qc.draw()`
- C. `qc.h(inp_reg[0:1])`
`qc.x(ancilla[0])`
`qc.draw()`
- ✓ D. `qc.h(inp_reg[0])`
`qc.h(inp_reg[1])`
`qc.x(ancilla[0])`
`qc.draw()`
- E. `qc.h(inp_reg[1])`
`qc.h(inp_reg[2])`
`qc.x(ancilla[1])`
`qc.draw()`
- F. `qc.h(inp_reg)`
`qc.h(inp_reg)`
`qc.x(ancilla)`
`qc.draw()`

Working through sample question #4

Familiarity with Qiskit API, measure vs. measure_all

4. Given an empty QuantumCircuit object, qc, with three qubits and three classical bits, which one of these code fragments would create this circuit?



- ✓ A. `qc.measure([0,1,2], [0,1,2])`
B. `qc.measure([0,0], [1,1], [2,2])`
C. `qc.measure_all()`
D. `qc.measure(0,1,2)`

```
measure(qubit, cbit)
```

Measure quantum bit into classical bit (tuples).

Parameters

- **qubit** (*QuantumRegister/list/tuple*) – quantum register
- **cbit** (*ClassicalRegister/list/tuple*) – classical register

```
measure_all(inplace=True) [SOURCE]
```

Adds measurement to all qubits. **Creates a new ClassicalRegister** with a size equal to the number of qubits being measured.

Returns a new circuit with measurements if *inplace=False*.

Parameters

inplace (*bool*) – All measurements inplace or return new circuit.

<https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html>

Another One

11. Which two options would place a barrier across all qubits to the QuantumCircuit below?

```
qc = QuantumCircuit(3,3)
```

- A. `qc.barrier(qc)`
- ✓ B. `qc.barrier([0,1,2])`
- ✓ C. `qc.barrier()`
- D. `qc.barrier(3)`
- E. `qc.barrier_all()`



```
barrier(*qargs)
```

Apply `Barrier`.

<https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html>

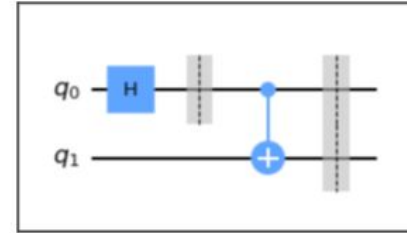
Anotha one

13. Given the following code, what is the depth of the circuit?

```
qc = QuantumCircuit(2, 2)

qc.h(0)
qc.barrier(0)
qc.cx(0,1)
qc.barrier([0,1])
```

- ✓ A. 2
- B. 3
- C. 4
- D. 5



`depth()` [SOURCE]

Return circuit depth (i.e., length of critical path). This does not include compiler or simulator directives such as 'barrier' or 'snapshot'.

Returns

Depth of circuit.

Return type

int

Nutha one

18. Which code fragment would yield an operator that represents a single-qubit X gate?

- A. `op = Operator.Xop(0)`
- B. `op = Operator([[0,1]])`
- ✓ C. `qc = QuantumCircuit(1)`
`qc.x(0)`
`op = Operator(qc)`
- D. `op = Operator([[1,0,0,1]])`

1.1 The X-Gate

The X-gate is represented by the Pauli-X matrix:

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

6.1 The I-gate

First comes the I-gate (aka 'Id-gate' or 'Identity gate'). This is simply a gate that does nothing. Its matrix is the identity matrix:

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

qiskit.quantum_info.Operator

CLASS `Operator`(*data*, *input_dims=None*, *output_dims=None*) [SOURCE]

Matrix operator class

This represents a matrix operator M that will `evolve()` a `Statevector` $|\psi\rangle$ by matrix-vector multiplication

$$|\psi\rangle \mapsto M|\psi\rangle,$$

and will `evolve()` a `DensityMatrix` ρ by left and right multiplication

$$\rho \mapsto M\rho M^\dagger.$$

Initialize an operator object.

Parameters

- **(QuantumCircuit or (data))** – Instruction or BaseOperator or matrix): data to initialize operator.
- **input_dims (tuple)** – the input subsystem dimensions. [Default: None]

https://qiskit.org/documentation/stubs/qiskit.quantum_info.Operator.html

Use single-qubit gates

```
circuit.h(qreg_q[0])
```

```
circuit_name.gate_name(register[index])
```

- Don't forget that the index starts at 0 for quantum and classical registers (like it often does in Python)



Use multi-qubit gates

Quantum computers use entanglement to separate themselves from classical ones, and one of the main ways to do this is via multi-qubit gates like the CNOT (also stylized as the CX).

CX has two qubits, but other gates, like the Toffoli, can have multiple controls (CCX) or multiple operations (CXX), but many quantum computers don't have CXX in their gate libraries, rather they do several CX in a row.

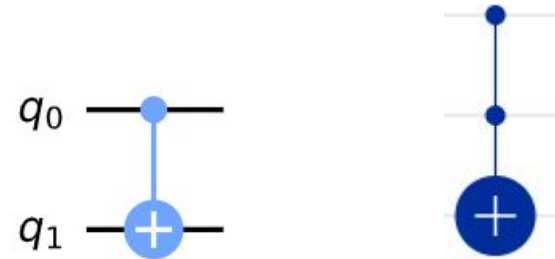


Use multi-qubit gates (CCX)

```
circuit.ccx(qreg_q[0], qreg_q[1], qreg_q[2])
```

Bold = controls

Non-bold = operation



CCX: The Toffoli gate, also known as the double controlled-NOT gate (CCX), has two control qubits and one target. It applies a NOT to the target only when both controls are in state $|1\rangle|1\rangle$.

The Toffoli gate with the Hadamard gate is a universal gate set for quantum computing.

Wait what is a universal gate set?

QuEra: A Universal Gate Set refers to a collection of quantum gates that can be used to approximate any unitary transformation on a quantum computer to an arbitrary degree of accuracy.

In classical computing, logical operations can be performed using a set of basic gates like AND, OR, and NOT. [**NAND** is often considered a universal gate, meaning any other gate can be made from NANDS together].

Similarly, in quantum computing, a universal gate set allows for the construction of **any quantum operation**, making it a foundational concept in quantum computation.

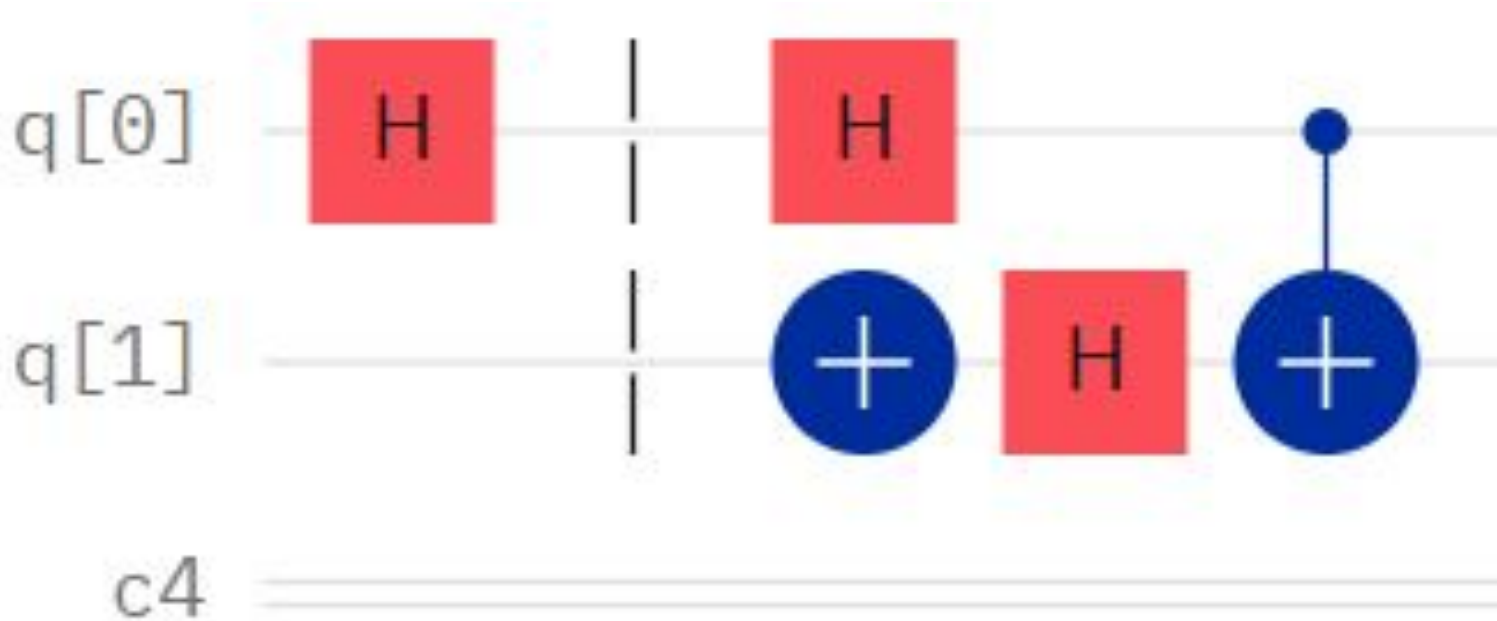


Use barrier operations

- What are barriers?
 - We do not actually have barriers in quantum computers, but they are used to isolate gates so that all of the gates do not “clump to the left”
 - A barrier is a visual indicator of the grouping of a circuit section. It also acts as a directive for circuit compilation to separate pieces of a circuit so that any optimizations or re-writes are constrained to only act between barriers.

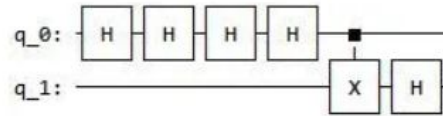


Use barrier operations



Return the circuit depth

```
print("Circuit depth: ", qc.depth())
```

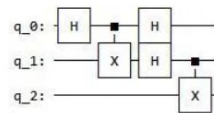


In this case, the first qubit suffers the action of 4 Hadamards and then a CX: depth 5.

The second qubit, apparently, only suffers the action of CX and then H. However, as it has to wait for the first qubit (depth 5) before applying H, in fact, the depth is 6.

Computing Circuit Depth

Example 3: The best way to do the math is to consider each port at once. For each qubit affected by the gate, consider which has the longest path. It is the Dynamic Programming technique.



The circuit starts with depth
[0 0 0]

Applying each gate:

H 0: [1 0 0]

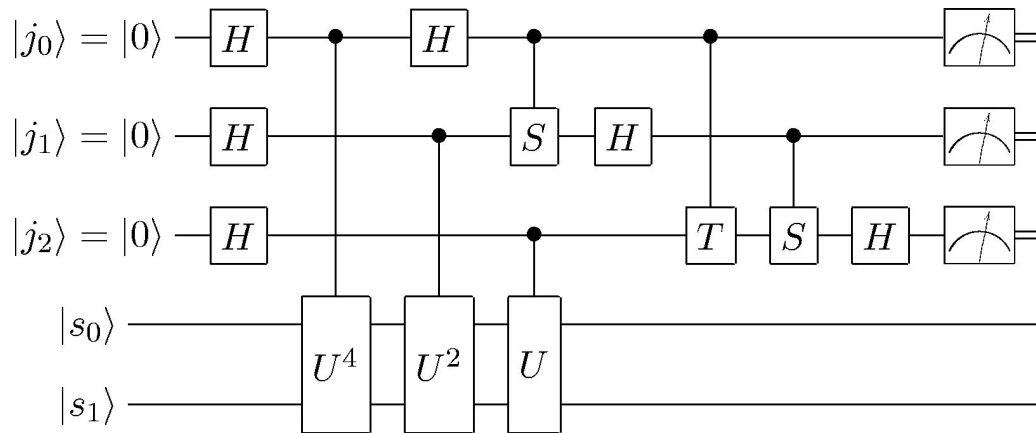
CX 0 1: [2 2 0] (Because CX affects 0 and 1, so you have to take the longest time between the two and add a unit)

H 0: [3 2 0]

H 1: [3 3 0]

CX 1 2: [3 4 4] (Because CX affects 1 and 2, so you have to take the longest time between the two and add a unit)

Computing the Circuit Depth



| | |
|------------------------|-----------------|
| H 1: | [1, 0, 0, 0, 0] |
| H 2: | [1, 1, 0, 0, 0] |
| H 3: | [1, 1, 1, 0, 0] |
| CU ⁴ 1 4 5: | [2, 1, 1, 2, 2] |
| CU ² 2 4 5: | [2, 3, 1, 3, 3] |
| CU ¹ 3 4 5: | [2, 2, 4, 4, 4] |
| H 1: | [3, 2, 4, 4, 4] |
| CS 1 2: | [4, 4, 4, 4, 4] |
| H 2: | [4, 5, 4, 4, 4] |
| CT 1 3: | [5, 5, 5, 4, 4] |
| CS 2 3: | [5, 6, 6, 4, 4] |
| H 3: | [5, 6, 7, 4, 4] |
| Meas 1: | [6, 6, 7, 4, 4] |
| Meas 2: | [6, 7, 7, 4, 4] |
| Meas 3: | [6, 7, 8, 4, 4] |

Extend quantum circuits

- It is hard to tell what this wants us to study, but a good example is on the IBMQ website, which we will look at on this page:

<https://docs.quantum.ibm.com/build/circuit-construction>



Return the OpenQASM string for a circuit

```
print(qc.qasm())
```

- This is literally all you have to do to return the string
- An example is on the following page



Return the OpenQASM string for a circuit

```
from qiskit import QuantumRegister, ClassicalRegister,  
QuantumCircuit  
from numpy import pi
```

```
qreg_q = QuantumRegister(2, 'q')  
creg_c = ClassicalRegister(4, 'c')  
circuit = QuantumCircuit(qreg_q, creg_c)
```

```
circuit.x(qreg_q[0])  
circuit.h(qreg_q[0])  
circuit.cx(qreg_q[0], qreg_q[1])  
circuit.s(qreg_q[0])  
circuit.t(qreg_q[1])  
circuit.rz(pi / 2, qreg_q[0])  
circuit.measure(qreg_q[0], creg_c[0])  
# @columns [0,1,2,3,3,4,5]
```

```
OPENQASM 2.0;  
include "qelib1.inc";
```

```
qreg q[2];  
creg c[4];
```

```
x q[0];  
h q[0];  
cx q[0], q[1];  
s q[0];  
t q[1];  
rz(pi/2) q[0];  
measure q[0] -> c[0];
```

```
// @columns [0,1,2,3,3,4,5]
```



OpenQASM on Composer

Increment and Decrement Saved File Edit View Visualizations seed 7694 Setup and run

Operations Freeform alignment Inspect

Search

q[0] q[1] c4

Amplitude

Computational basis states

Statevector

Output state

Qiskit

Open in Quantum Lab

```
1 from qiskit import
  QuantumRegister,
  ClassicalRegister,
  QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(2,
  'q')
5 creg_c = ClassicalRegister
  (4, 'c')
6 circuit = QuantumCircuit
  (qreg_q, creg_c)
7
8 circuit.x(qreg_q[0])
9 circuit.h(qreg_q[0])
10 circuit.cx(qreg_q[0], qreg_q
  [1])
11 circuit.s(qreg_q[0])
12 circuit.t(qreg_q[1])
13 circuit.rz(pi / 2, qreg_q[0])
14 circuit.measure(qreg_q[0],
  creg_c[0])
15 # @columns [0,1,2,3,3,4,5]
```

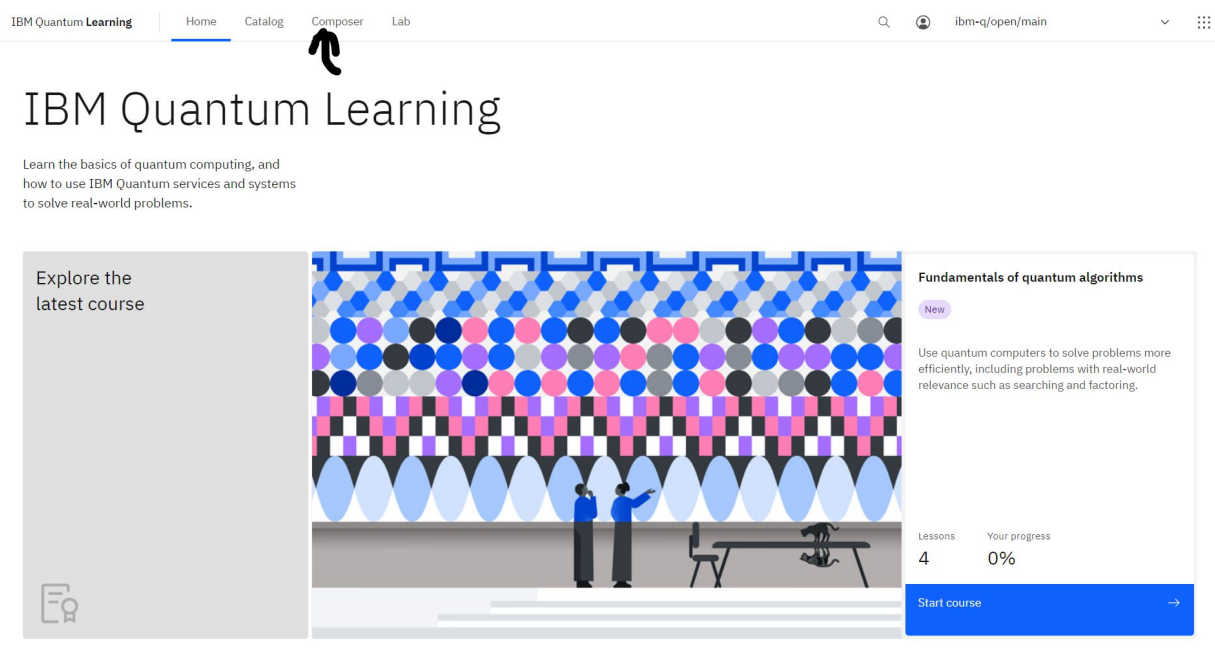
More on the Composer

I don't think we can use this on the exam but all of the study resources say it is the best so here it is...



Accessing the Composer

<https://learning.quantum.ibm.com/>



What does it look like?

IBM Quantum Learning | Home | Catalog | **Composer** | Lab

Search | ibm-q/open/main

My first circuit *Saved* | File | Edit | View | Visualizations seed 7694 | Setup and run

Operations

Search

q[0]
q[1]
c4

Freeform alignment | Inspect

OpenQASM 2.0

Open in Quantum Lab

```
1 OPENQASM 2.0;  
2 include "qelib1.inc";  
3  
4 qreg q[2];  
5 creg c[4];  
6  
7 // @columns []  
8
```

Probabilities

Probability (%)

Computational basis states

Q-sphere

State ☒ Phase angle ☐

ATM

View Tab

IBM Quantum Learning | Home | Catalog | **Composer** | Lab

Search | ibm-q/open/main | Visualizations seed 7694 | Setup and run

My first circuit *Saved* | **File** | Edit | **View**

Operations | Freeform alignment | Inspect

Search | q[0] | q[1] | c4

Operations palette: H, \oplus , \otimes , \otimes^2 , \otimes^3 , I, T, S, Z, T^\dagger , S^\dagger , P, RZ, \sqrt{X} , $|0\rangle$, $|1\rangle$, if, \sqrt{X}^\dagger , Y, RX, RY, RXX, RZZ, U, RCCX, RC3X, $\frac{\pi}{4}$

Probabilities | Q-sphere

Probability (%) | Computational basis states

Q-sphere | State ☒ | Phase angle ☐

OpenQASM 2.0 | Open in Quantum Lab

```
1 OPENQASM 2.0;
2 include "qelib1.inc";
3
4 qreg q[2];
5 creg c[4];
6
7 // @columns []
8
```

Terms | Privacy | Cookie preferences | Support

ATM

What can it do?

Visualize qubit states

See how changes to your circuit affect the state of qubits, shown as an interactive q-sphere, or histograms showing measurement probabilities/statevector simulations.

Run on quantum hardware

Run your circuits on real quantum hardware to understand the effects of device noise.

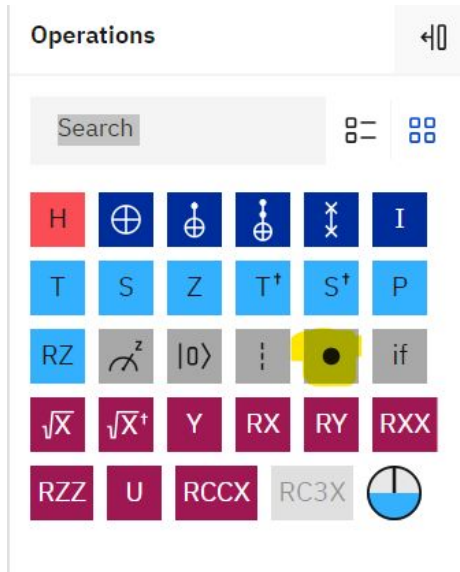
Automatically generate code

Instead of writing code by hand, automatically generate OpenQASM or Python code that behaves the same way as the circuit you created with Quantum Composer.



Can add controls

- One of the interesting things you can do is add controls



Phase Disks

Phase disk

A single-qubit state can be represented as

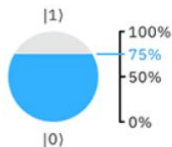
$$|\psi\rangle = \sqrt{1-p}|0\rangle + e^{j\varphi}\sqrt{p}|1\rangle,$$

where p is the probability that the qubit is in the $|1\rangle$ state, and φ is the quantum phase. p is strongly analogous to a classical probabilistic bit. For $p = 0$, the qubit is in the $|0\rangle$ state, for $p = 1$ the qubit is in the $|1\rangle$ state, and for $p = 1/2$ the qubit is a 50/50 mixture. We call this a superposition as, unlike classical bits, this mixture can have a quantum phase. The phase disk visualizes this state.

The phase disk at the terminus of each qubit in IBM Quantum Composer gives the local state of each qubit at the end of the computation. The components of the phase disk are described below.

Probability the qubit is in the $|1\rangle$ state

The probability that the qubit is in the $|1\rangle$ state is represented by the blue disk fill.

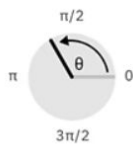


Quantum phase

Phase Disks

Quantum phase

The quantum phase of the qubit state is given by the line that extends from the center of the diagram to the edge of the gray disk (which rotates counterclockwise around the center point).



Example: phase disks for two different qubits



∠ Phase θ : 0

$$\begin{array}{|l} \text{Re}[e^{j\theta}]: 0 \\ \text{Im}[e^{j\theta}]: 0 \end{array}$$

● Prob of $|1\rangle$: 100%

○ Purity of reduced state: 1



∠ Phase θ : π

$$\begin{array}{|l} \text{Re}[e^{j\theta}]: 0 \\ \text{Im}[e^{j\theta}]: 0 \end{array}$$

● Prob of $|1\rangle$: 50%

○ Purity of reduced state: 0.5

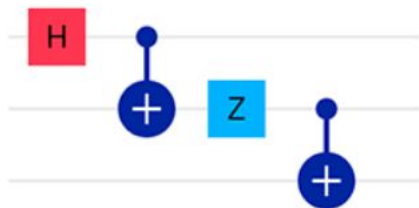
Create a custom operation in OpenQASM

You can define new unitary operations in the code editor (see the figure below for an example). The operations are applied using the statement `name(params) qargs;` just like the built-in operations. The parentheses are optional if there are no parameters.

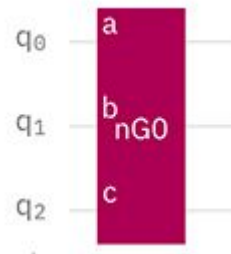
To define a custom operation, enter it in the OpenQASM code editor using this format: `gate name(params) qargs;`. If you click **+Add** in the operations list, you will be prompted to enter a name for your custom operation, which you can then build in the code editor.

Once you have defined your custom operation, drag it onto the graphical editor and use the edit icon to fine-tune the parameters.

a) The gates to be included in the custom operation:



```
gate nG0 a, b, c {  
  h a;  
  cx a, b;  
  z b;  
  cx b, c;  
}
```



Gates Recap



Don't beef with me I just can't do it better than these visuals from the operation glossary can

<https://learning.quantum.ibm.com/tutorial/explore-gates-and-circuits-with-the-quantum-composer#composer-operations-glossary>

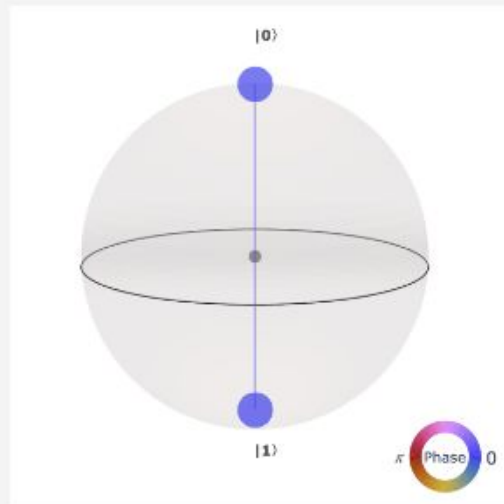


NOT Gate

The NOT gate, also known as the Pauli X gate, flips the $|0\rangle$ state to $|1\rangle$, and vice versa. The NOT gate is equivalent to RX for the angle π or to 'HZH'.



x q[0];



Note about q-sphere representations

The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$, where n is the number of qubits needed to support the gate.

Composer
reference

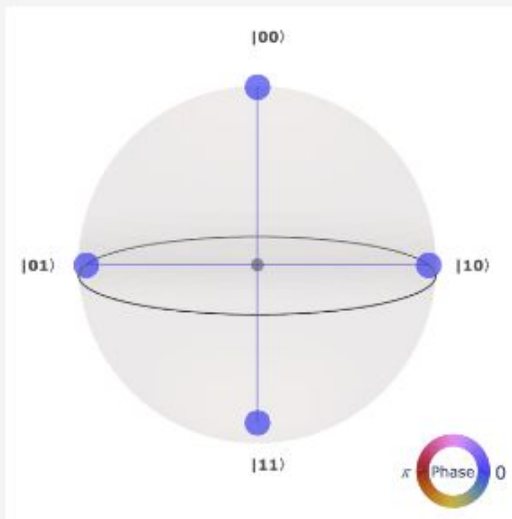
OpenQASM
reference

Q-Sphere

Note about q-sphere representations



```
cx q[0],  
q[1];
```



The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$, where n is the number of qubits needed to support the gate.

Composer
reference

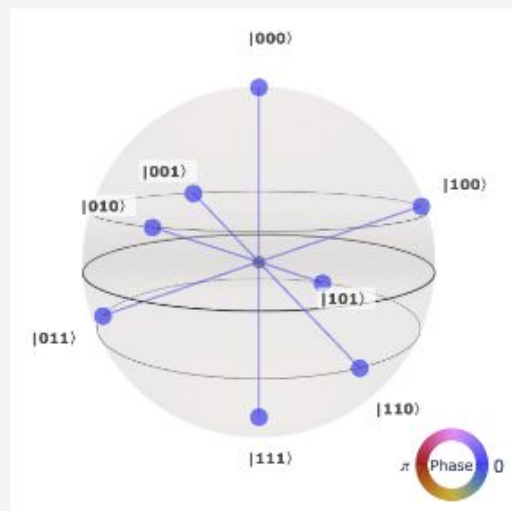
OpenQASM
reference

Q-Sphere

Note about q-sphere representations



```
ccx q[0],  
q[1],  
q[2];
```



The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$, where n is the number of qubits needed to support the gate.

Composer
reference

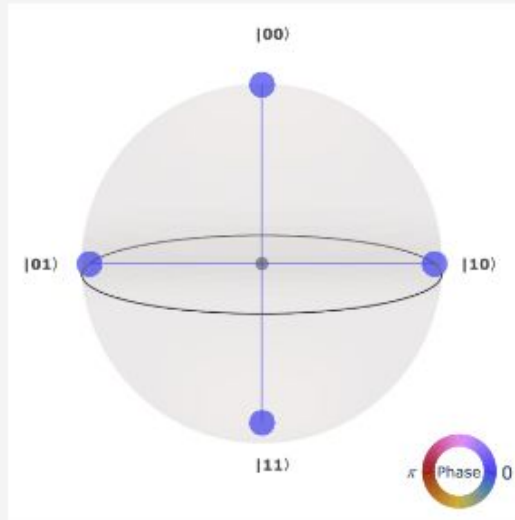
OpenQASM
reference

Q-Sphere

Note about q-sphere representations




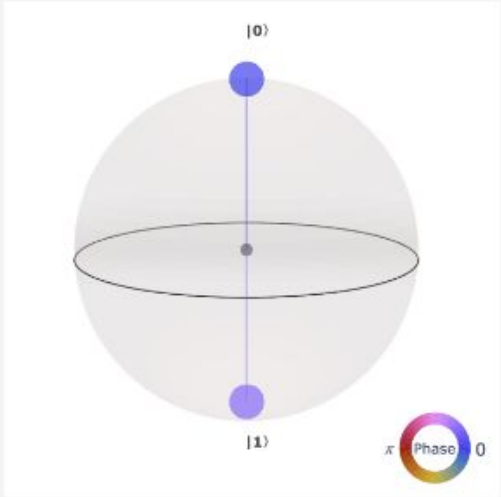
```
swap q[0],  
q[1];
```



The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$, where n is the number of qubits needed to support the gate.

T gate

The T gate is equivalent to RZ for the angle $\pi/4$. Fault-tolerant quantum computers will compile all quantum programs down to just the T gate and its inverse, as well as the Clifford gates.

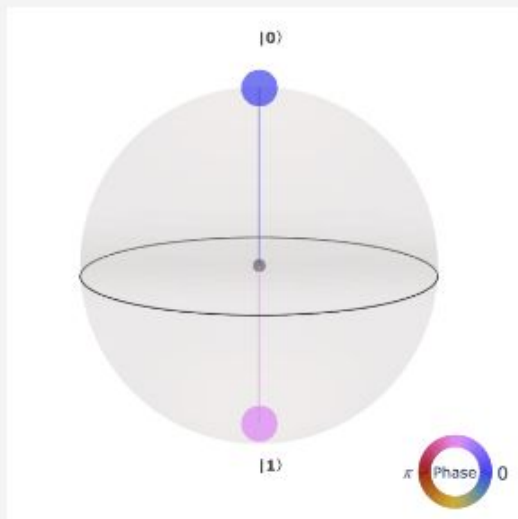
| Composer reference | OpenQASM reference | Q-Sphere | Note about q-sphere representations |
|---|----------------------|--|--|
|  | <code>t q[0];</code> |  | The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} i\rangle$, where n is the number of qubits needed to support the gate. |

S gate

The S gate applies a phase of i to the $|1\rangle$ state. It is equivalent to RZ for the angle $\pi/2$. Note that $S=P(\pi/2)$.



s q[0];


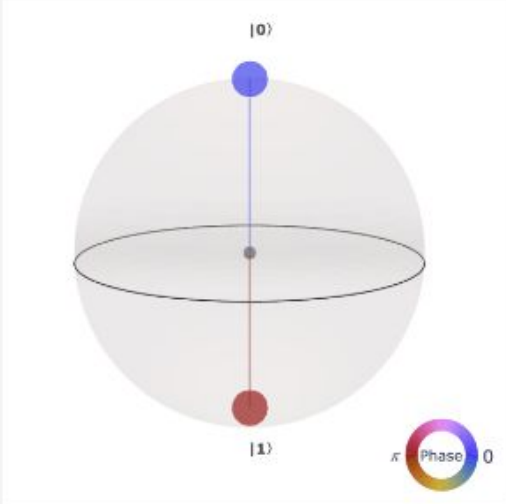


Note about q-sphere representations

The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$, where n is the number of qubits needed to support the gate.


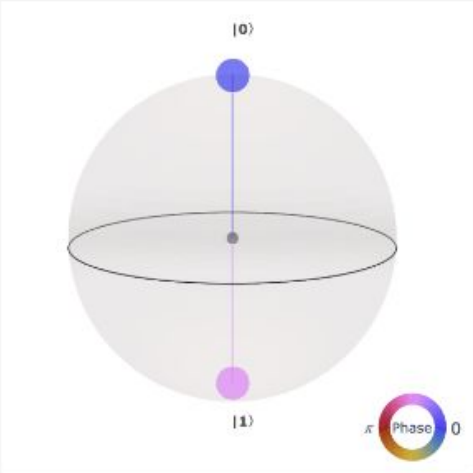
Z gate

The Pauli Z gate acts as identity on the $|0\rangle$ state and multiplies the sign of the $|1\rangle$ state by -1. It therefore flips the $|+\rangle$ and $|-\rangle$ states. In the $+/-$ basis, it plays the same role as the NOT gate in the $|0\rangle/|1\rangle$ basis.

| Composer reference | OpenQASM reference | Q-Sphere | Note about q-sphere representations |
|---|--------------------|--|--|
|  | <pre>z q[0];</pre> |  | The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} i\rangle$, where n is the number of qubits needed to support the gate. |

Phase gate

The Phase gate (previously called the U1 gate) applies a phase of $e^{i\theta}$ to the $|1\rangle$ state. For certain values of θ , it is equivalent to other gates. For example, $P(\pi)=Z$, $P(\pi/2)=S$, and $P(\pi/4)=T$. Up to a global phase of $e^{i\theta/2}$, it is equivalent to $RZ(\theta)$.

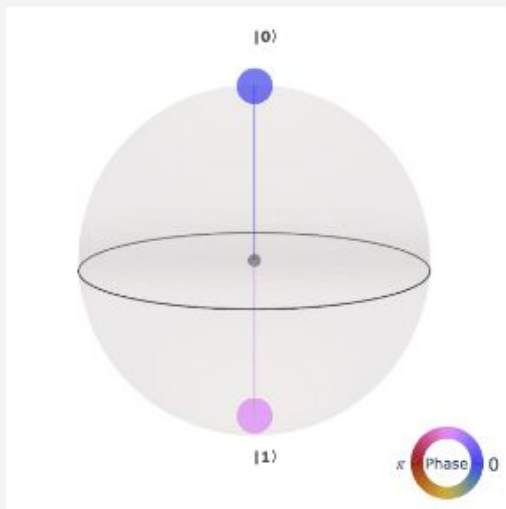
| Composer reference | OpenQASM reference | Q-Sphere | Note about q-sphere representations |
|---|-------------------------------|--|--|
|  | <pre>p(theta) q[0];</pre> |  | The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} i\rangle$, where n is the number of qubits needed to support the gate. |

RZ gate

The RZ gate implements $\exp(-i\frac{\theta}{2}Z)$. On the Bloch sphere, this gate corresponds to rotating the qubit state around the z axis by the given angle.



```
rz(angle)  
q[0];
```



Note about q-sphere representations

The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$, where n is the number of qubits needed to support the gate.

Non-unitary operators and modifiers


Reset operation

The reset operation returns a qubit to state $|0\rangle$, irrespective of its state before the operation was applied. It is not a reversible operation.

| Composer reference | OpenQASM reference |
|---|--------------------------|
|  | <code>reset q[0];</code> |

Measurement


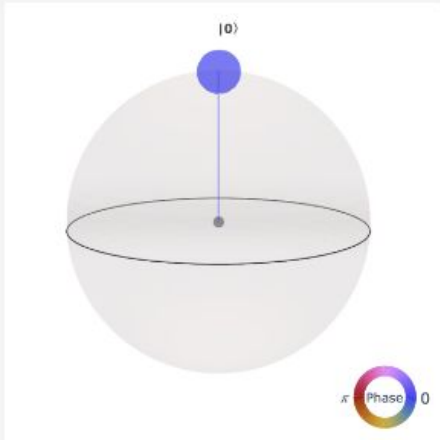
Measurement in the standard basis, also known as the z basis or computational basis. Can be used to implement any kind of measurement when combined with gates. It is not a reversible operation.

| Composer reference | OpenQASM reference |
|---|----------------------------|
|  | <code>measure q[0];</code> |

Hadamard gate

H gate

The H, or Hadamard, gate rotates the states $|0\rangle$ and $|1\rangle$ to $|+\rangle$ and $|-\rangle$, respectively. It is useful for making superpositions. If you have a universal gate set on a classical computer and add the Hadamard gate, it becomes a universal gate set on a quantum computer.

| Composer reference | OpenQASM reference | Q-Sphere | Note about q-sphere representations |
|--|----------------------|---|--|
|  | <code>h q[0];</code> |  | The q-sphere representation shows the state after the gate operates on the initial equal superposition state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} i\rangle$, where n is the number of qubits needed to support the gate. |

Any Questions?



Planning for next meeting

MARCH 2024

| SUN | MON | TUE | WED | THU | FRI | SAT |
|----------|-----|-----|-----|-----|-----|-----|
| 25 31 | 26 | 27 | 28 | 29 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | | | | | | |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |

www.GrabCalendar.com



April Planning

APRIL 2024

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | | 25 | 26 | 27 |
| 28 | 29 | | | | | |

www.GrabCalendar.com



Homework?

In no particular order:

Lessons (0/4 complete)

Collapse all lessons

Single systems

Introduction

Classical information

Quantum information

Qiskit examples

Go to lesson →

Multiple systems

Introduction

Classical information

Quantum information

Qiskit examples

Go to lesson →

Quantum circuits

Introduction

Circuits

Inner products, orthonormality, and projections

Limitations on quantum information

Go to lesson →

Entanglement in action

Introduction

Teleportation

Superdense coding

The CHSH game

Go to lesson →

Exam

Take this exam to test your skills. This exam is intended to be taken after reading the lessons in this course. Once you have completed the exam, come back here to see your earned badge.

Take the exam →

Awarded badge

Basics of Quantum Information

IBM Quantum Foundation

IBM leverages the services of Credly, a 3rd party data processor authorized by IBM and located in the United States, to assist in the administration of the IBM Digital Badge program. In order to issue you an IBM Digital Badge, your personal information (name, email address, and badge earned) will be shared with Credly.

You will receive an email notification from Credly with instructions for claiming the badge. Your personal information is used to issue your badge and for program reporting and operational purposes. IBM may share the personal information collected with IBM subsidiaries and third parties globally. It will be handled in a manner consistent with [IBM Privacy Statement](#).



Homework?

Certification



IBM Certified Associate
Developer - Quantum
Computation using Qiskit v0.2X

Register for exam →

An **Assessment Exam** is an online test that results in a score report to help you gauge your preparedness. They can be booked through Pearson VUE.

[Assessment Exam →](#)

The **Sample Test** is designed to give you an idea of the type of questions you can expect to see on the exam.

[Sample Test →](#)

The **Study Guide** provides additional detail and references for the exam objectives.

[Study Guide PDF →](#)



The End

