



- 
- 
- Documentation
- Community
- Learn
- Overview


# Learn Quantum Computation using Qiskit

- Learn Quantum Computation using Qiskit
- What is Quantum?
- 0. Prerequisites
  - 0.1 Setting Up Your Environment
  - 0.2 Python and Jupyter Notebooks
- 1. Quantum States and Qubits
  - 1.1 Introduction
  - 1.2 The Atoms of Computation
  - 1.3 Representing Qubit States
  - 1.4 Single Qubit Gates
  - 1.5 The Case for Quantum
- 2. Multiple Qubits and Entanglement
  - 2.1 Introduction
  - 2.2 Multiple Qubits and Entangled States
  - 2.3 Phase Kickback
  - 2.4 More Circuit Identities
  - 2.5 Proving Universality
  - 2.6 Classical Computation on a Quantum Computer
- 3. Quantum Protocols and Quantum Algorithms
  - 3.1 Defining Quantum Circuits
  - 3.2 Deutsch-Jozsa Algorithm
  - 3.3 Bernstein-Vazirani Algorithm
  - 3.4 Simon's Algorithm
  - 3.5 Quantum Fourier Transform
  - 3.6 Quantum Phase Estimation
  - 3.7 Shor's Algorithm
  - 3.8 Grover's Algorithm
  - 3.9 Quantum Counting
  - 3.10 Quantum Walk Search Algorithm
  - 3.11 Quantum Teleportation
  - 3.12 Superdense Coding
  - 3.13 Quantum Key Distribution
- 4. Quantum Algorithms for Applications

- 4.1 Applied Quantum Algorithms
  - 4.1.1 Solving Linear Systems of Equations using HHL
  - 4.1.2 Simulating Molecules using VQE
  - 4.1.3 Solving combinatorial optimization problems using QAOA
  - 4.1.4 Solving Satisfiability Problems using Grover's Algorithm
  - 4.1.5 Hybrid quantum-classical Neural Networks with PyTorch and Qiskit
- 4.2 Implementations of Recent Quantum Algorithms
  - 4.2.1 Variational Quantum Linear Solver
  - 4.2.2 Quantum Image Processing - FRQI and NEQR Image Representations
  - 4.2.3 Quantum Edge Detection - QHED Algorithm on Small and Large Images
  - 4.2.4 Solving the Travelling Salesman Problem using Phase Estimation
- 5. Investigating Quantum Hardware Using Quantum Circuits
  - 5.1 Introduction to Quantum Error Correction using Repetition Codes
  - 5.2 Measurement Error Mitigation
  - 5.3 Randomized Benchmarking
  - 5.4 Measuring Quantum Volume
  - 5.5 The Density Matrix & Mixed States
- 6. Investigating Quantum Hardware Using Microwave Pulses
  - 6.1 Calibrating Qubits with Qiskit Pulse
  - 6.2 Accessing Higher Energy States
  - 6.3 Introduction to Transmon Physics
  - 6.4 Circuit Quantum Electrodynamics
  - 6.5 Exploring the Jaynes-Cummings Hamiltonian with Qiskit Pulse
  - 6.6 Measuring the Qubit ac-Stark Shift
  - 6.7 Hamiltonian Tomography
- 7. Quantum Computing Labs
  - Lab 1. Quantum Circuits
  - Lab 2. Quantum Measurement
  - Lab 3. Accuracy of Quantum Phase Estimation
  - Lab 4. Iterative Quantum Phase Estimation
  - Lab 5. Scalable Shor's Algorithm
  - Lab 6. Grover's search with an unknown number of solutions
  - Lab 7. Quantum Simulation as a Search Algorithm
  - Lab 8. Quantum Error Correction
- 8. Appendix
  - 8.1 Linear Algebra
  - 8.2 Qiskit
- 9. Games & Demos
  - Hello Qiskit Game
  - Estimating Pi Using Quantum Phase Estimation Algorithm
  - Local Reality and the CHSH Inequality
  - Quantum Coin Game
  - Variational Quantum Regression
  - Interactivity Index

## On This Page

- Contents
- 1. Classical vs Quantum Bits
  - 1.1 Statevectors
  - 1.2 Qubit Notation
  - 1.3 Exploring Qubits with Qiskit
- 2. The Rules of Measurement
  - 2.1 A Very Important Rule
  - 2.2 The Implications of this Rule
  - #1 Normalisation
  - #2 Alternative measurement
  - #3 Global Phase
  - #4 The Observer Effect
  - A Note about Quantum Simulators
- 3. The Bloch Sphere
  - 3.1 Describing the Restricted Qubit State
  - 3.2 Visually Representing a Qubit State
  - Version Information

[Open in IBM Quantum Lab](#)[Download as Jupyter Notebook](#) [Contribute on Github](#) [Try the new textbook beta](#) 

The new Qiskit Textbook beta is now available. Try it out now

## Representing Qubit States

You now know something about bits, and about how our familiar digital computers work. All the complex variables, objects and data structures used in modern software are basically all just big piles of bits. Those of us who work on quantum computing call these *classical variables*. The computers that use them, like the one you are using to read this article, we call *classical computers*.

In quantum computers, our basic variable is the *qubit*: a quantum variant of the bit. These have exactly the same restrictions as normal bits do: they can store only a single binary piece of information, and can only ever give us an output of 0 or 1. However, they can also be manipulated in ways that can only be described by quantum mechanics. This gives us new gates to play with, allowing us to find new ways to design algorithms.

To fully understand these new gates, we first need to understand how to write down qubit states. For this we will use the mathematics of vectors, matrices, and complex numbers. Though we will introduce these concepts as we go, it would be best if you are comfortable with them already. If you need a more in-depth explanation or a refresher, you can find the guide [here](#).

## Contents


- 1. Classical vs Quantum Bits
  - 1.1 Statevectors

- 1.2 Qubit Notation
  - 1.3 Exploring Qubits with Qiskit
- 2. The Rules of Measurement
  - 2.1 A Very Important Rule
  - 2.2 The Implications of this Rule
- 3. The Bloch Sphere
  - 3.1 Describing the Restricted Qubit State
  - 3.2 Visually Representing a Qubit State

# 1. Classical vs Quantum Bits


## 1.1 Statevectors

In quantum physics we use *statevectors* to describe the state of our system. Say we wanted to describe the position of a car along a track, this is a classical system so we could use a number  $x$ :

 tracking a car with scalars

$$x = 4$$

Alternatively, we could instead use a collection of numbers in a vector called a *statevector*. Each element in the statevector contains the probability of finding the car in a certain place:

 tracking a car with vectors

$$|x\rangle = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \leftarrow \begin{array}{l} \text{Probability of} \\ \text{car being at} \\ \text{position 4} \end{array}$$

This isn't limited to position, we could also keep a statevector of all the possible speeds the car could have, and all the possible colours the car could be. With classical systems (like the car example above), this is a silly thing to do as it requires keeping huge vectors when we only really need one number. But as we will see in this chapter, statevectors happen to be a very good way of keeping track of quantum systems, including quantum computers.

## 1.2 Qubit Notation

Classical bits always have a completely well-defined state: they are either 0 or 1 at every point during a computation. There is no more detail we can add to the state of a bit than this. So to write down the state of a classical bit ( $c$ ), we can just use these two binary values. For example:

$$c = 0$$

This restriction is lifted for quantum bits. Whether we get a 0 or a 1 from a qubit only needs to be well-defined when a measurement is made to extract an output. At that point, it must commit to one of these two options. At all other times, its state will be something more complex than can be captured by a simple binary value. To see how to describe these, we can first focus on the two simplest cases. As we saw in the last section, it is possible to prepare a qubit in a state for which it definitely gives the outcome 0 when measured. We need a name for this state. Let's be unimaginative and call it 0 . Similarly, there exists a qubit state that is certain to output a 1. We'll call this 1. These two states are completely mutually exclusive. Either the qubit definitely outputs a 0, or it definitely outputs a 1. There is no overlap. One way to represent this with mathematics is to use two orthogonal vectors.

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

This is a lot of notation to take in all at once. First, let's unpack the weird  $|$  and  $\rangle$ . Their job is essentially just to remind us that we are talking about the vectors that represent qubit states labelled 0 and 1. This helps us distinguish them from things like the bit values 0 and 1 or the numbers 0 and 1. It is part of the bra-ket notation, introduced by Dirac. If you are not familiar with vectors, you can essentially just think of them as lists of numbers which we manipulate using certain rules. If you are familiar with vectors from your high school physics classes, you'll know that these rules make vectors well-suited for describing quantities with a magnitude and a direction. For example, the velocity of an object is described perfectly with a vector. However, the way we use vectors for quantum states is slightly different to this, so don't hold on too hard to your previous intuition. It's time to do something new! With vectors we can describe more complex states than just  $|0\rangle$  and  $|1\rangle$ . For example, consider the vector

$$|q_0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{bmatrix}.$$

To understand what this state means, we'll need to use the mathematical rules for manipulating vectors. Specifically, we'll need to understand how to add vectors together and how to multiply them by scalars.

- [Reminder: Matrix Addition and Multiplication by Scalars \(Click here to expand\)](#)
- [Reminder: Orthonormal Bases \(Click here to expand\)](#)

Since the states  $|0\rangle$  and  $|1\rangle$  form an orthonormal basis, we can represent any 2D vector with a combination of these two states. This allows us to write the state of our qubit in the alternative form:

$$|q_0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle$$

This vector,  $|q_0\rangle$  is called the qubit's *statevector*, it tells us everything we could possibly know about this qubit. For now, we are only able to draw a few simple conclusions about this particular example of a statevector: it is not entirely  $|0\rangle$  and not entirely  $|1\rangle$ . Instead, it is described by a linear combination of the two. In quantum mechanics, we typically describe linear combinations such as this using the word 'superposition'. Though our example state  $|q_0\rangle$  can be expressed as a superposition of  $|0\rangle$  and  $|1\rangle$ , it is no less a definite and well-defined qubit state than they are. To see this, we can begin to explore how a qubit can be manipulated.

### 1.3 Exploring Qubits with Qiskit

First, we need to import all the tools we will need:

```
from qiskit import QuantumCircuit, assemble, Aer
from qiskit.visualization import plot_histogram, plot_bloch_vector
from math import sqrt, pi
```



In Qiskit, we use the QuantumCircuit object to store our circuits, this is essentially a list of the quantum operations on our circuit and the qubits they are applied to.

```
qc = QuantumCircuit(1) # Create a quantum circuit with one qubit
```



In our quantum circuits, our qubits always start out in the state  $|0\rangle$ . We can use the `initialize()` method to transform this into any state. We give `initialize()` the vector we want in the form of a list, and tell it which qubit(s) we want to initialize in this state:

```
qc = QuantumCircuit(1) # Create a quantum circuit with one qubit
initial_state = [0,1] # Define initial_state as |1>
qc.initialize(initial_state, 0) # Apply initialisation operation to the 0th qubit
qc.draw() # Let's view our circuit
```



q —  $\begin{matrix} |\psi\rangle \\ [0,1] \end{matrix}$  —

We can then use one of Qiskit's simulators to view the resulting state of our qubit.

```
sim = Aer.get_backend('aer_simulator') # Tell Qiskit how to simulate our circuit
```



To get the results from our circuit, we use `run` to execute our circuit, giving the circuit and the backend as arguments. We then use `.result()` to get the result of this:

```
qc = QuantumCircuit(1) # Create a quantum circuit with one qubit
initial_state = [0,1] # Define initial_state as |1>
qc.initialize(initial_state, 0) # Apply initialisation operation to the 0th qubit
qc.save_statevector() # Tell simulator to save statevector
qobj = assemble(qc) # Create a Qobj from the circuit for the simulator to run
result = sim.run(qobj).result() # Do the simulation and return the result
```



from result, we can then get the final statevector using `.get_statevector()`:

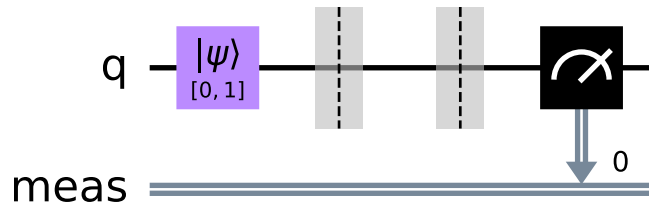
```
out_state = result.get_statevector()
print(out_state) # Display the output state vector
```



[0.+0.j 1.+0.j]

Note: Python uses  $j$  to represent  $i$  in complex numbers. We see a vector with two complex elements:  $0.+0.j = 0$ , and  $1.+0.j = 1$ . Let's now measure our qubit as we would in a real quantum computer and see the result:

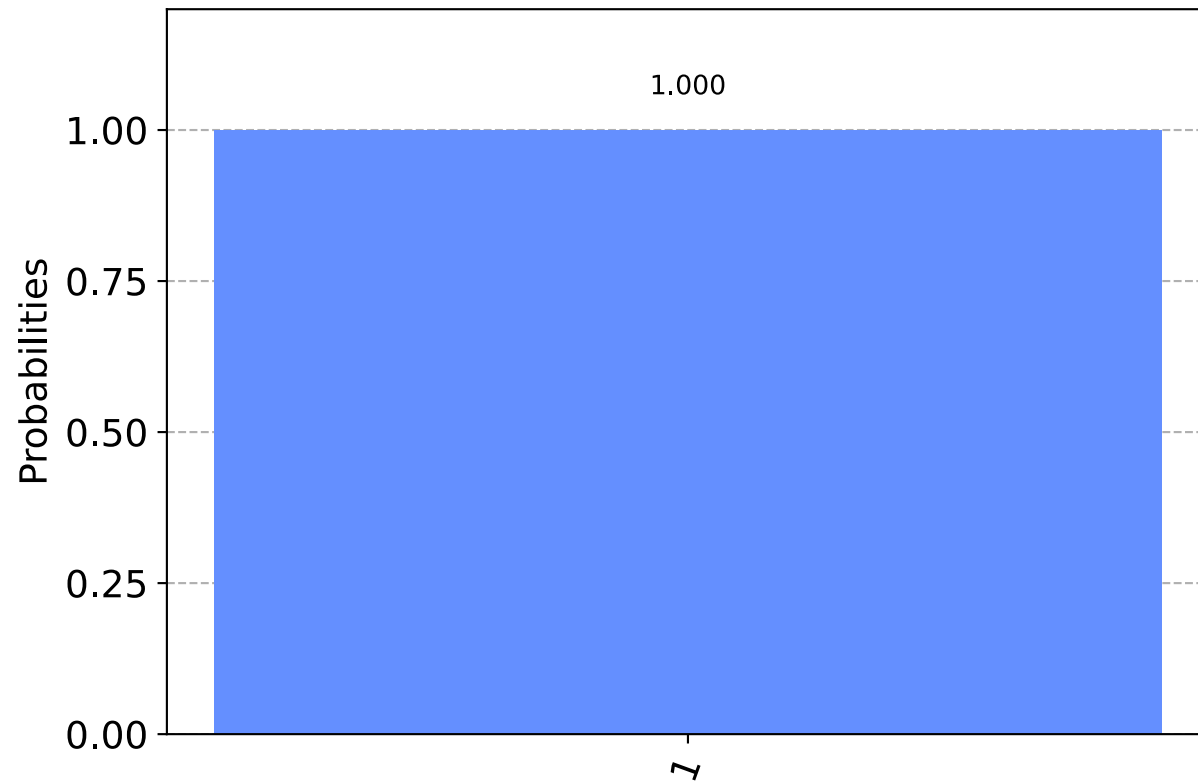
```
qc.measure_all()  
qc.draw()
```



This time, instead of the statevector we will get the counts for the 0 and 1 results using `.get_counts()`:

```
qobj = assemble(qc)  
result = sim.run(qobj).result()  
counts = result.get_counts()  
plot_histogram(counts)
```





We can see that we (unsurprisingly) have a 100% chance of measuring  $|1\rangle$ . This time, let's instead put our qubit into a superposition and see what happens. We will use the state  $|q_0\rangle$  from earlier in this section:

$$|q_0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle$$

We need to add these amplitudes to a python list. To add a complex amplitude, Python uses `j` for the imaginary unit (we normally call it "*i*" mathematically):

```
initial_state = [1/sqrt(2), 1j/sqrt(2)] # Define state |q_0>
```



And we then repeat the steps for initialising the qubit as before:

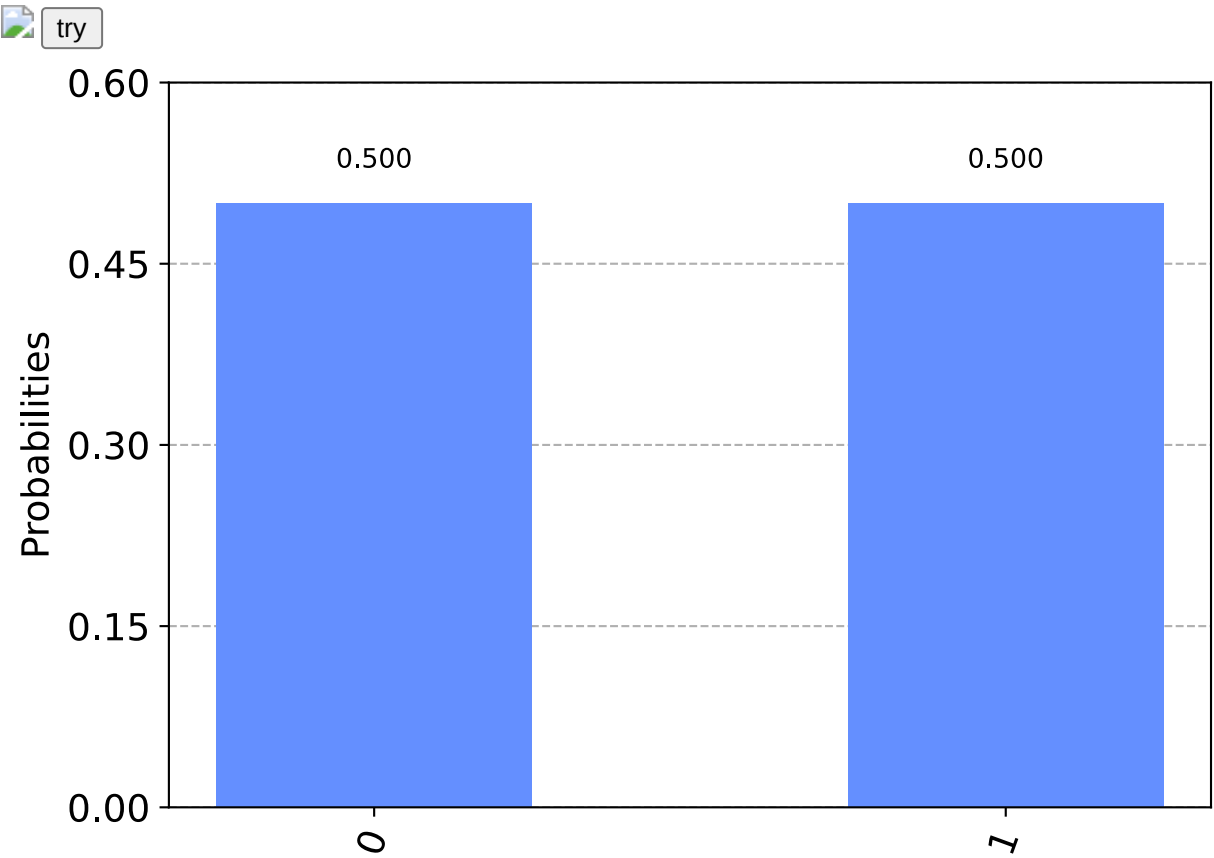
```
qc = QuantumCircuit(1) # Must redefine qc
qc.initialize(initial_state, 0) # Initialize the 0th qubit in the state `initial_state`
qc.save_statevector() # Save statevector
qobj = assemble(qc)
state = sim.run(qobj).result().get_statevector() # Execute the circuit
print(state) # Print the result
```



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



```
qobj = assemble(qc)
results = sim.run(qobj).result().get_counts()
plot_histogram(results)
```



We can see we have equal probability of measuring either  $|0\rangle$  or  $|1\rangle$ . To explain this, we need to talk about measurement.

## 2. The Rules of Measurement

### 2.1 A Very Important Rule

There is a simple rule for measurement. To find the probability of measuring a state  $|\psi\rangle$  in the state  $|x\rangle$  we do:

$$p(|x\rangle) = |\langle x|\psi\rangle|^2$$

The symbols  $\langle$  and  $|$  tell us  $\langle x|$  is a row vector and the symbols  $|$  and  $\rangle$  tell us  $|\psi\rangle$  is a column vector. In quantum mechanics we call the column vectors *kets* and the row vectors *bras*. Together they make up *bra-ket* notation. Any ket  $|a\rangle$  has a corresponding bra  $\langle a|$ , and we convert between them using the conjugate transpose.

- [Reminder: Conjugate Transpose \(Click here to expand\)](#)
- [Reminder: The Inner Product \(Click here to expand\)](#)

In the equation above,  $|x\rangle$  can be any qubit state. To find the probability of measuring  $|x\rangle$ , we take the inner product of  $|x\rangle$  and the state we are measuring (in this case  $|\psi\rangle$ ), then square the magnitude. This may seem a little convoluted, but it will soon become second nature. If we look at the state  $|q_0\rangle$  from before, we can see the probability of measuring  $|0\rangle$  is indeed 0.5:

$$\begin{aligned} |q_0\rangle &= \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle \\ \langle 0|q_0\rangle &= \frac{1}{\sqrt{2}}\langle 0|0\rangle + \frac{i}{\sqrt{2}}\langle 0|1\rangle \\ &= \frac{1}{\sqrt{2}} \cdot 1 + \frac{i}{\sqrt{2}} \cdot 0 \\ &= \frac{1}{\sqrt{2}} \\ |\langle 0|q_0\rangle|^2 &= \frac{1}{2} \end{aligned}$$

You should verify the probability of measuring  $|1\rangle$  as an exercise. This rule governs how we get information out of quantum states. It is therefore very important for everything we do in quantum computation. It also immediately implies several important facts.

## 2.2 The Implications of this Rule

### #1 Normalisation

The rule shows us that amplitudes are related to probabilities. If we want the probabilities to add up to 1 (which they should!), we need to ensure that the statevector is properly normalized. Specifically, we need the magnitude of the state vector to be 1.

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

Thus if:

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

Then:

$$|\alpha|^2 + |\beta|^2 = 1$$

This explains the factors of  $\sqrt{2}$  you have seen throughout this chapter. In fact, if we try to give `initialize()` a vector that isn't normalised, it will give us an error:

```
vector = [1,1]
qc.initialize(vector, 0)
```



```
-----
QiskitError                                Traceback (most recent call last)
<ipython-input-12-ddc73828b990> in <module>
```

```

1 vector = [1,1]
----> 2 qc.initialize(vector, 0)

/usr/local/anaconda3/lib/python3.7/site-packages/qiskit/extensions/quantum_initializer/initializer.py in initialize(self, params, qubits)
    453
    454     num_qubits = None if not isinstance(params, int) else len(qubits)
--> 455     return self.append(Initialize(params, num_qubits), qubits)
    456
    457

/usr/local/anaconda3/lib/python3.7/site-packages/qiskit/extensions/quantum_initializer/initializer.py in __init__(self, params, num_qubits)
    89         if not math.isclose(sum(np.absolute(params) ** 2), 1.0,
    90                               abs_tol=_EPS):
--> 91             raise QiskitError("Sum of amplitudes-squared does not equal one.")
    92
    93         num_qubits = int(num_qubits)

```

QiskitError: 'Sum of amplitudes-squared does not equal one.'

## Quick Exercise

1. Create a state vector that will give a  $1/3$  probability of measuring  $|0\rangle$ .
2. Create a different state vector that will give the same measurement probabilities.
3. Verify that the probability of measuring  $|1\rangle$  for these two states is  $2/3$ .

You can check your answer in the widget below (accepts answers  $\pm 1\%$  accuracy, you can use numpy terms such as 'pi' and 'sqrt()' in the vector):

```

# Run the code in this cell to interact with the widget
from qiskit_textbook.widgets import state_vector_exercise
state_vector_exercise(target=1/3)

```



try

## #2 Alternative measurement

The measurement rule gives us the probability  $p(|x\rangle)$  that a state  $|\psi\rangle$  is measured as  $|x\rangle$ . Nowhere does it tell us that  $|x\rangle$  can only be either  $|0\rangle$  or  $|1\rangle$ .

The measurements we have considered so far are in fact only one of an infinite number of possible ways to measure a qubit. For any orthogonal pair of states, we can define a measurement that would cause a qubit to choose between the two.

This possibility will be explored more in the next section. For now, just bear in mind that  $|x\rangle$  is not limited to being simply  $|0\rangle$  or  $|1\rangle$ .

## #3 Global Phase

We know that measuring the state  $|1\rangle$  will give us the output 1 with certainty. But we are also able to write down states such as

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

To see how this behaves, we apply the measurement rule.

$$|\langle x|(i|1\rangle)|^2 = |i\langle x|1\rangle|^2 = |\langle x|1\rangle|^2$$

Here we find that the factor of  $i$  disappears once we take the magnitude of the complex number. This effect is completely independent of the measured state  $|x\rangle$ . It does not matter what measurement we are considering, the probabilities for the state  $i|1\rangle$  are identical to those for  $|1\rangle$ . Since measurements are the only way we can extract any information from a qubit, this implies that these two states are equivalent in all ways that are physically relevant.

More generally, we refer to any overall factor  $\gamma$  on a state for which  $|\gamma| = 1$  as a 'global phase'. States that differ only by a global phase are physically indistinguishable.

$$|\langle x|(\gamma|a\rangle)|^2 = |\gamma\langle x|a\rangle|^2 = |\langle x|a\rangle|^2$$

Note that this is distinct from the phase difference *between* terms in a superposition, which is known as the 'relative phase'. This becomes relevant once we consider different types of measurement and multiple qubits.

## #4 The Observer Effect

We know that the amplitudes contain information about the probability of us finding the qubit in a specific state, but once we have measured the qubit, we know with certainty what the state of the qubit is. For example, if we measure a qubit in the state:

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

And find it in the state  $|0\rangle$ , if we measure again, there is a 100% chance of finding the qubit in the state  $|0\rangle$ . This means the act of measuring *changes* the state of our qubits.

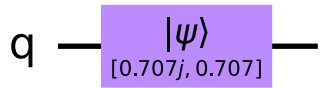
$$|q\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \xrightarrow{\text{Measure } |0\rangle} |q\rangle = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

We sometimes refer to this as *collapsing* the state of the qubit. It is a potent effect, and so one that must be used wisely. For example, were we to constantly measure each of our qubits to keep track of their value at each point in a computation, they would always simply be in a well-defined state of either  $|0\rangle$  or  $|1\rangle$ . As such, they would be no different from classical bits and our computation could be easily replaced by a classical computation. To achieve truly quantum computation we must allow the qubits to explore more complex states. Measurements are therefore only used when we need to extract an output. This means that we often place all the measurements at the end of our quantum circuit.

We can demonstrate this using Qiskit's statevector simulator. Let's initialize a qubit in superposition:

```
qc = QuantumCircuit(1) # We are redefining qc
initial_state = [0.+1.j/sqrt(2),1/sqrt(2)+0.j]
qc.initialize(initial_state, 0)
qc.draw()
```





This should initialize our qubit in the state:

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

We can verify this using the simulator:

```
qc.save_statevector()
result = sim.run(assemble(qc)).result()
state = result.get_statevector()
print("Qubit State = " + str(state))
```

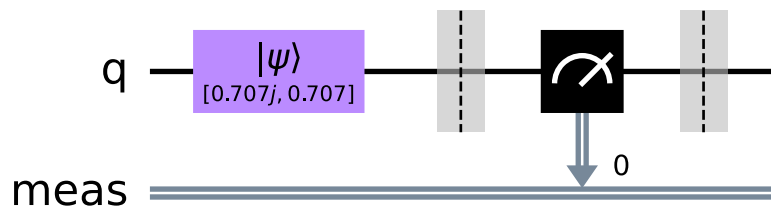


Qubit State = [0. +0.70710678j 0.70710678+0.j]

We can see here the qubit is initialized in the state  $[0. +0.70710678j \ 0.70710678+0.j]$ , which is the state we expected.

Let's now create a circuit where we measure this qubit:

```
qc = QuantumCircuit(1) # We are redefining qc
initial_state = [0.+1.j/sqrt(2),1/sqrt(2)+0.j]
qc.initialize(initial_state, 0)
qc.measure_all()
qc.save_statevector()
qc.draw()
```



When we simulate this entire circuit, we can see that one of the amplitudes is *a*lways 0:

```
qobj = assemble(qc)
state = sim.run(qobj).result().get_statevector()
```

```
print("State of Measured Qubit = " +str(state))
```



```
State of Measured Qubit = [0.+1.j 0.+0.j]
```

You can re-run this cell a few times to reinitialize the qubit and measure it again. You will notice that either outcome is equally probable, but that the state of the qubit is never a superposition of  $|0\rangle$  and  $|1\rangle$ . Somewhat interestingly, the global phase on the state  $|0\rangle$  survives, but since this is global phase, we can never measure it on a real quantum computer.

## A Note about Quantum Simulators

We can see that writing down a qubit's state requires keeping track of two complex numbers, but when using a real quantum computer we will only ever receive a yes-or-no (0 or 1) answer for each qubit. The output of a 10-qubit quantum computer will look like this:

```
0110111110
```

Just 10 bits, no superposition or complex amplitudes. When using a real quantum computer, we cannot see the states of our qubits mid-computation, as this would destroy them! This behaviour is not ideal for learning, so Qiskit provides different quantum simulators: By default, the `aer_simulator` mimics the execution of a real quantum computer, but will also allow you to peek at quantum states before measurement if we include certain instructions in our circuit. For example, here we have included the instruction `.save_statevector()`, which means we can use `.get_statevector()` on the result of the simulation.

## 3. The Bloch Sphere

### 3.1 Describing the Restricted Qubit State

We saw earlier in this chapter that the general state of a qubit ( $|q\rangle$ ) is:

$$|q\rangle = \alpha|0\rangle + \beta|1\rangle$$
$$\alpha, \beta \in \mathbb{C}$$

(The second line tells us  $\alpha$  and  $\beta$  are complex numbers). The first two implications in section 2 tell us that we cannot differentiate between some of these states. This means we can be more specific in our description of the qubit.

Firstly, since we cannot measure global phase, we can only measure the difference in phase between the states  $|0\rangle$  and  $|1\rangle$ . Instead of having  $\alpha$  and  $\beta$  be complex, we can confine them to the real numbers and add a term to tell us the relative phase between them:

$$|q\rangle = \alpha|0\rangle + e^{i\phi}\beta|1\rangle$$
$$\alpha, \beta, \phi \in \mathbb{R}$$

Finally, since the qubit state must be normalised, i.e.

$$\sqrt{\alpha^2 + \beta^2} = 1$$

we can use the trigonometric identity:

$$\sqrt{\sin^2 x + \cos^2 x} = 1$$

to describe the real  $\alpha$  and  $\beta$  in terms of one variable,  $\theta$ :

$$\alpha = \cos \frac{\theta}{2}, \quad \beta = \sin \frac{\theta}{2}$$

From this we can describe the state of any qubit using the two variables  $\phi$  and  $\theta$ :

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

$$\theta, \phi \in \mathbb{R}$$

## 3.2 Visually Representing a Qubit State

We want to plot our general qubit state:

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

If we interpret  $\theta$  and  $\phi$  as spherical co-ordinates ( $r = 1$ , since the magnitude of the qubit state is 1), we can plot any single qubit state on the surface of a sphere, known as the *Bloch sphere*.

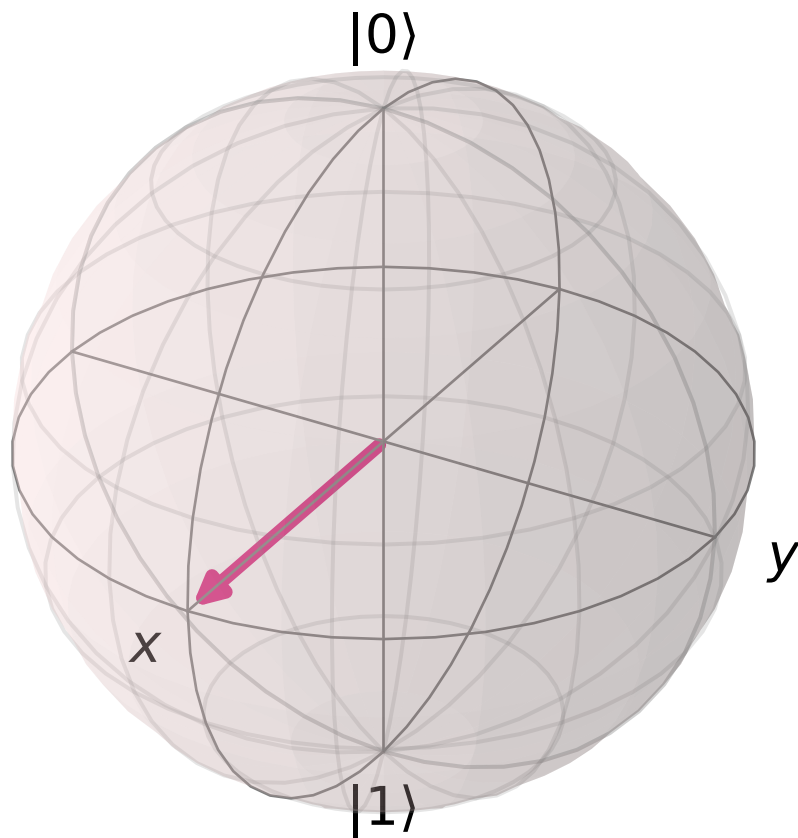
Below we have plotted a qubit in the state  $|+\rangle$ . In this case,  $\theta = \pi/2$  and  $\phi = 0$ .

(Qiskit has a function to plot a bloch sphere, `plot_bloch_vector()`, but at the time of writing it only takes cartesian coordinates. We have included a function that does the conversion automatically).

You can also try this interactive Bloch sphere demo.

```
from qiskit_textbook.widgets import plot_bloch_vector_spherical
coords = [pi/2,0,1] # [Theta, Phi, Radius]
plot_bloch_vector_spherical(coords) # Bloch Vector with spherical coordinates
```





### Warning!

When first learning about qubit states, it's easy to confuse the qubits *statevector* with its *Bloch vector*. Remember the statevector is the vector discussed in 1.1, that holds the amplitudes for the two states our qubit can be in. The Bloch vector is a visualisation tool that maps the 2D, complex statevector onto real, 3D space.

### Quick Exercise

Use `plot_bloch_vector()` or `plot_bloch_vector_spherical()` to plot a qubit in the states:

1.  $|0\rangle$
2.  $|1\rangle$
3.  $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
4.  $\frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$



5.  $\frac{1}{\sqrt{2}} \begin{bmatrix} i \\ 1 \end{bmatrix}$

We have also included below a widget that converts from spherical co-ordinates to cartesian, for use with `plot_bloch_vector()`:

```
from qiskit_textbook.widgets import bloch_calc
bloch_calc()
```



```
import qiskit.tools.jupyter
%qiskit_version_table
```



### Version Information

Qiskit Software	Version
Qiskit	0.27.0
Terra	0.17.4
Aer	0.8.2
Ignis	0.6.0
Aqua	0.9.2
IBM Q Provider	0.14.0
<b>System information</b>	
Python	3.7.7 (default, May 6 2020, 04:59:01) [Clang 4.0.1 (tags/RELEASE_401/final)]
OS	Darwin
CPUs	8
Memory (Gb)	32.0

Wed Jun 16 13:17:34 2021 BST

〈 The Atoms of Computation Single Qubit Gates 〉

This page was created by The Jupyter Book Community