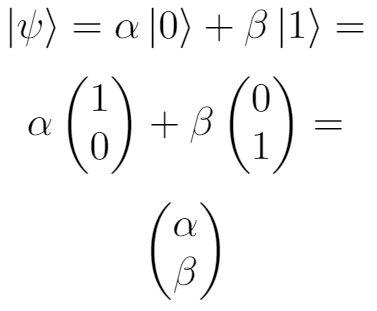
**Introduction to Quantum logic gates**

One of the most important parts of a quantum computer and the building blocks of every quantum algorithm.



Photo by [Fractal Hassan](https://unsplash.com/@tetromino?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

In our previous article [Quantum computing — The big picture](https://medium.com/wtm-algiers-we-write/quantum-computing-the-big-picture-2c61bf80dbe1), we discovered the possible states a qubit could take. We saw that qubits could be represented as a *superposition* of two states with 2D vectors:

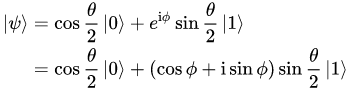


where:

α = cosθ/2

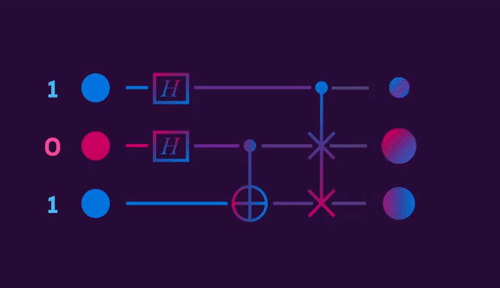
β= e^iϕsinθ/2

θ and ϕ are real numbers.



In this article we will cover *gates,* ***the operations that change a qubit between these states***.

**What is a Quantum circuit?**

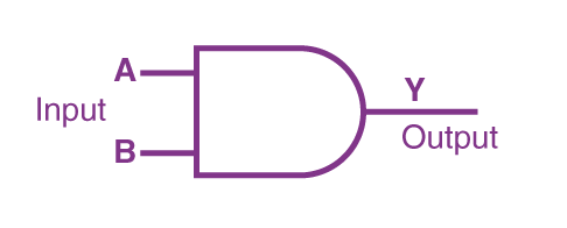


Quantum Circuit — Animation from Dinge erklärt Kurzgesagt

Similar to [classical circuits](https://en.wikipedia.org/wiki/Circuit_(computer_science)), a quantum circuit is an ordered sequence of *quantum gates,*[*measurements*](https://en.wikipedia.org/wiki/Measurement_in_quantum_mechanics)*and initializations of qubits to known values*. Any quantum program can be represented by a sequence of quantum circuits and non-concurrent classical computation.

**What is a logic gate?**

A basic electronic circuit that operates on one or more inputs to produce an output. In other words, when data pass through the gate, it changes its state, and this change depends on the input and the gate itself.

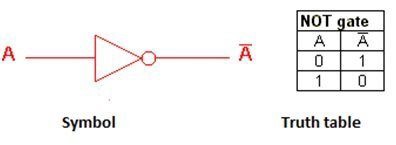


Simple representation of a logic gate

**Classical logic gates**

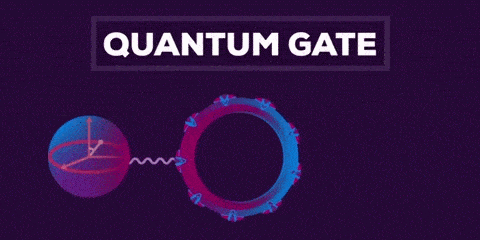
Classical logic gates are the fundamental operations of a classical computer, allowing bits to change their states between 1 and 0. We have already mentioned that the change depends on the input and the type of the gate. There are multiple examples of classical logic gates such as: NOT, AND, OR, NAND, NOR, XOR, etc.

For example, a NOT gate changes a bit from a 0 to a 1 (or vice versa). AND, and OR gates are two-bit gates that take two bits as inputs and output a single bit, depending on the inputs.



NOT gate

**Quantum logic gates**



Quantum Gate — Animation from Dinge erklärt Kurzgesagt

Quantum computers operate using [qubits](https://medium.com/@universalquantum/why-every-qubit-is-not-created-equal-93c20433cc70), not bits. Unlike traditional bits which can only be 0 **or** 1, a qubit can exist in a ‘superposition’ of 0 **and** 1.



Superposition — Animation from Dinge erklärt Kurzgesagt

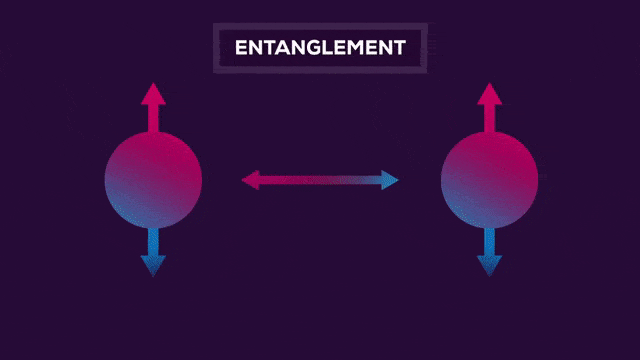
What makes a qubit so powerful is its ability to carry out quantum calculations which cannot be achieved without the fundamental operations, known as **quantum logic gates**.

The most common quantum gates operate on spaces of one or two qubits. This means that as matrices, quantum gates can be described by 2 x 2 or 4 x 4 matrices with [orthonormal](https://www.quantiki.org/search/node/orthonormal) rows.



Orthogonal matrix

*There are lots of types of quantum gates. There are****single-qubit gates****, which can flip a qubit from 0 to 1 as well as allowing superposition states to be created. Then there are also****two-qubit gates****. These allow the qubits to interact with each other and can be used to create quantum*entanglement*: a state of two or more qubits that are correlated.*



**Quantum gates are reversible!**

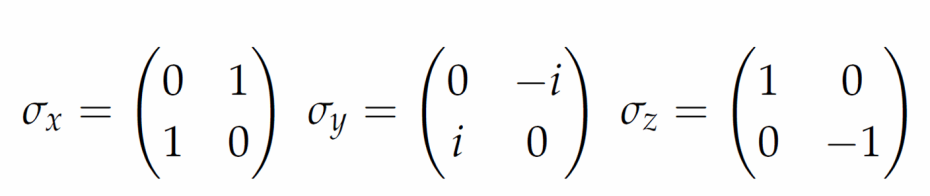
An important feature of quantum circuits is that the operations (gates) are *always* [reversible](https://www.ijraset.com/research-paper/introduction-to-reversible-logic-gates-and-its-operations)! These reversible gates can be represented as matrices, and as rotations around the Bloch sphere.

from qiskit import QuantumCircuit, assemble, Aer  
from math import pi, sqrt  
from qiskit.visualization import plot\_bloch\_multivector, plot\_histogram  
sim **=** Aer**.**get\_backend('aer\_simulator')

**The Pauli Gates**

**Reminder: Pauli matrices**

In physics, the **Pauli matrices** are a set of 2 × 2 [complex](https://en.wikiversity.org/wiki/Complex_number) [Hermitian](https://en.wikiversity.org/w/index.php?title=Hermitian_matrix&action=edit&redlink=1) and [unitary](https://en.wikiversity.org/w/index.php?title=Unitary_matrix&action=edit&redlink=1) [matrices](https://en.wikiversity.org/w/index.php?title=Matrix_(mathematics)&action=edit&redlink=1). Usually indicated by the Greek letter “sigma” (σ)

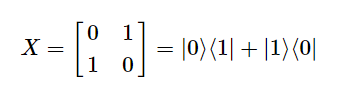


Mathematical representation of the Pauli X, Y & Z matrices

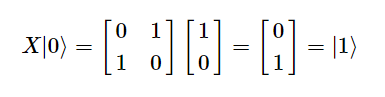
We will see here that the Pauli matrices can represent some very commonly used quantum gates.

**The Pauli X-Gate**

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

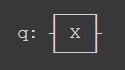


We can change a state of a qubit by multiplying its state-vector by the gate. We can see that the X-gate switches the amplitudes of the states |0⟩ and |1⟩:



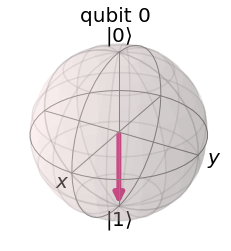
In Qiskit, we can create a short circuit to verify this:

*# Let's apply a Pauli X-gate on a |0> qubit*  
qc **=** QuantumCircuit(1) #create a new circuit with one register  
qc**.**x(0) #create a |0> qubit  
qc**.**draw()



To visualize the result of our quantum circuit, we will need to use plot\_bloch\_multivector() which takes a qubit's *state-vector*instead of the *Bloch vector*.

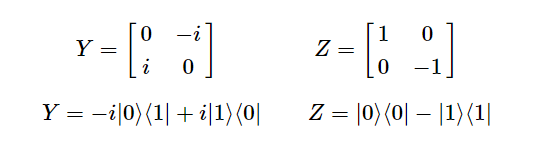
*# Let's visualize the result*  
qc**.**save\_statevector()  
qobj **=** assemble(qc)  
state **=** sim**.**run(qobj)**.**result()**.**get\_statevector()  
plot\_bloch\_multivector(state)



We can indeed see the state of the qubit is |1⟩ as expected. We can think of this as a rotation by**π radians** around the *x-axis* of the Bloch sphere. Similarly to the classical NOT gate, the X-gate changes the state from |0> to |1> (vice versa), that’s why it is also often called a NOT-gate.

**The Pauli Y & Z-gates**

The Y & Z Pauli matrices in the previous illustration, also act as the Y & Z-gates in our quantum circuits:

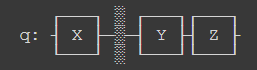


Pauli Y & Z Gates

In the Bloch sphere, they respectively represent rotations by π around the y and z-axis.

In Qiskit, we can apply the Y and Z-gates to our circuit using:

qc**.**y(0) *# Do Y-gate on qubit 0*  
qc**.**z(0) *# Do Z-gate on qubit 0*  
qc**.**draw()



**The X, Y & Z-Bases**

**Reminder: Eigenvectors of Matrices**

We have seen that multiplying a vector |v> by a matrix M results in a vector |v’>:

M|v⟩=|v′⟩ ←new vector

There is a special case where multiplying a vector |v> by a matrix M is equivalent to multiplying the same matrix M by a scalar λ.

M|v⟩=λ|v⟩

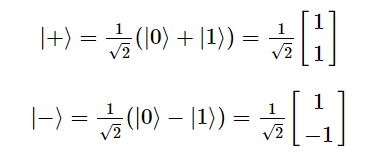
Any vector |v> that has this property, is called an *Eigenvector* of the matrix M.

Consider the following example, where:

Z|0⟩=|0⟩

Z|1⟩=−|1⟩

From the results, we conclude that the qubit states |0> and |1> are Eigenvectors of the Pauli Z-matrix. Since our qubits states are represented by 2D vectors, we often call these vectors *Eigenstates.*The basis formed by the states |0⟩ and |1⟩ is called the **Z-basis**. In fact, it is not the only basis we can use, we also have the **X-basis**, formed by the eigenstates of the**Pauli X-gate**. We call these two vectors |+⟩ and |−⟩:

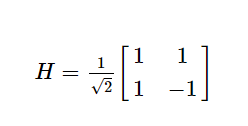


The X-basis

Pauli-gates alone are not sufficient to move the qubit to any state other than |0⟩ or |1⟩, which means we can never achieve superposition that differentiates between a qubit and a classical bit. To create more interesting states, we will need more gates like:

**The Hadamard Gate**

The Hadamard gate (H-gate) is a fundamental quantum gate. It allows us to move away from the poles of the Bloch sphere and create a superposition of |0⟩ and |1⟩. It has the matrix:



We can see that this performs the transformations below:

H|0⟩=|+⟩

H|1⟩=|−⟩

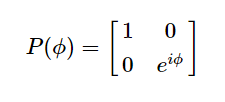
This can be thought of as transforming the state of the qubit between the X and Z bases.

You can test with these gates using the widget below:

*# Run the code in this cell to see the widget*  
from qiskit\_textbook.widgets import gate\_demo  
gate\_demo(gates**=**'pauli+h')

**The P-gate**

The P-gate (phase gate) is *parametrised.*It performs a rotation of ϕ around the Z-axis direction. It has the matrix form:



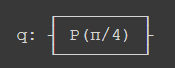
Where ϕ is a real number.

You can use the widget below to play around with the P-gate, specify ϕ using the slider:

*# Run the code in this cell to see the widget*  
from qiskit\_textbook.widgets import gate\_demo  
gate\_demo(gates**=**'pauli+h+p')

In Qiskit, we specify a P-gate using p(phi, qubit):

qc **=** QuantumCircuit(1)  
qc**.**p(pi**/**4, 0)  
qc**.**draw()

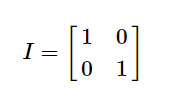


*Tip: The Pauli Z-gate is a special case of the P-gate, with ϕ=π. Actually, there are three more common gates that are special cases of the P-gate: I, S and T-Gates.*

**The I, S and T-gates**

**1. The I-gate**

First comes the I-gate (aka ‘Id-gate’ or ‘Identity gate’) which does nothing. It ‘s represented by the identity matrix:



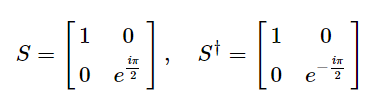
Applying the identity gate anywhere in your circuit should have no effect on the qubit state. There are two main use cases:

* In calculations, for example: proving the X-gate is its own inverse: I=XI=X
* When we want real hardware to specify a ‘do-nothing’ or ‘none’ operation.

**2. The S-gates**

Sometimes known as the √Z-gate, it is a P-gate with **ϕ=π/2**. It does a **quarter-turn** around the Bloch sphere.

Unlike the previous gates, the S-gate is **not** its own inverse! That’s why, you will often see the S†-gate, (also “S-dagger”, “Sdg” or √Z†-gate). The S†-gate is clearly a P-gate with ϕ=−π/2:



The name “√Z-gate” comes from the fact that two successively applied S-gates has the same effect as one Z-gate:

SS|q⟩=Z|q⟩

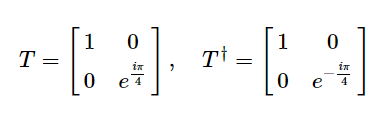
To apply an S-gate on our qubit in Qiskit:

qc **=** QuantumCircuit(1)  
qc**.**s(0) *# Apply S-gate to qubit 0*  
qc**.**sdg(0) *# Apply Sdg-gate to qubit 0*  
qc**.**draw()



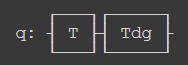
**3. The T-gate**

The T-gate is a P-gate with **ϕ=π/4**:



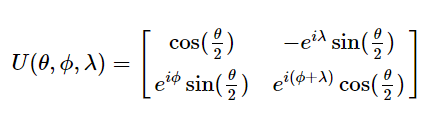
As with the S-gate, the T-gate is sometimes also known as the 4√Z-gate.

qc **=** QuantumCircuit(1)  
qc**.**t(0) *# Apply T-gate to qubit 0*  
qc**.**tdg(0) *# Apply Tdg-gate to qubit 0*  
qc**.**draw()



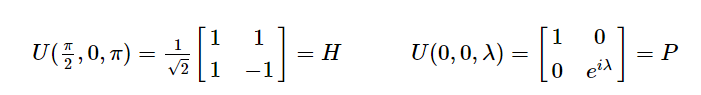
**The U-gate**

As we saw earlier, the I, Z, S & T-gates were all special cases of the more general P-gate. In the same way, the U-gate is the most general of all single-qubit quantum gates. It is a parametrised gate of the form:



Every gate in this chapter could be specified as U(θ,ϕ,λ), but it is unusual to see this in a circuit diagram, possibly due to the difficulty in reading this.

As an example, we see some specific cases of the U-gate in which it is equivalent to the **H-gate** and **P-gate** respectively.

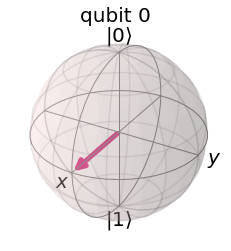


H-Gate and P-Gate are special cases of the U-Gate

*# Let's have U-gate transform a |0> to |+> state*  
qc **=** QuantumCircuit(1)  
qc**.**u(pi**/**2, 0, pi, 0)  
qc**.**draw()



*# Let's see the result*  
qc**.**save\_statevector()  
qobj **=** assemble(qc)  
state **=** sim**.**run(qobj)**.**result()**.**get\_statevector()  
plot\_bloch\_multivector(state)



**Resources**

You can find a community-created cheat-sheet with some of the common quantum gates, and their properties [here](https://raw.githubusercontent.com/qiskit-community/qiskit-textbook/main/content/ch-states/supplements/single-gates-cheatsheet.pdf).

Link to the whole [code](https://github.com/ihssene-nasa/Qiskit_tutorial/blob/main/Qiskit_Quantum_Gates.ipynb)

ThanQu for reading!

*Note: This article is a summary of what we learned from the*[*Qiskit*](https://qiskit.org/textbook/preface.html)*course*

*Written with ❤ By Geminae Stellae (*

[*Ihssene Brahimi*](https://medium.com/u/14952b38d6b9?source=post_page-----543785b7f252--------------------------------)

*&*

[*Assala benmalek*](https://medium.com/u/459235e3bb31?source=post_page-----543785b7f252--------------------------------)

*)*

272

[More from Geminae Stellae 💫](https://medium.com/@geminae.stellae?source=post_page-----543785b7f252--------------------------------)

Follow

Quantum computing | Data science | Parallel computing | Personal experiences of Assala Benmalek & Ihssene Brahimi