CPEN 400Q Lecture 02
Quantum circuits and PennyLane

Friday 13 January 2023

Announcements

- Assignment 0 due on Monday; Assignment 1 next week
- First quiz on Monday; contents from Monday and today's lectures

Last time

We outlined the structure of quantum algorithms:

1. **Prepare** qubits in a **superposition**
2. Apply **operations** that **entangle** the qubits and manipulate the amplitudes
3. **Measure** qubits to extract an answer

Last time

Qubits are physical quantum systems with two **basis states**:

States are written as complex vectors in **Hilbert space**.

Arbitrary states are linear combinations of the basis states:

where $|\alpha|^2 + |\beta|^2 = 1$ and $\alpha, \beta \in \mathbb{C}$.

Last time

Unitary matrices (gates/operations) modify a qubit's state.

A matrix U is unitary if

$$UU^\dagger = U^\dagger U = \mathbb{1}.$$

They preserve lengths of state vectors and angles between them.

Some examples:

Last time

Measurement at the end of an algorithm is probabilistic.

If we measure a qubit in state

we observe it in

- $|0\rangle$ with probability
- $|1\rangle$ with probability

Last time

We wrote some NumPy code to do all this:

```
def ket_0():
    return np.array([1, 0])

def ket_1():
    return np.array([0, 1])


def superposition(alpha, beta):
    return alpha * ket_() + beta * ket_1()


def apply_op(U, state):
    return np.dot(U, state)

def apply_ops(list_U, state):
    for U in list_U:
        state = np.dot(U, state)
    return state
```

Last time

```python
def measure(state, num_samples):
    prob_0 = np.abs(state[0]) ** 2
    prob_1 = state[1] * state[1].conj()

    samples = np.random.choice(
        [0, 1], size=num_samples, p=[prob_0, prob_1]
    )

    return samples
```

```python
def quantum_algorithm(alpha, beta, list_U):
    initial_state = superposition(alpha, beta)
    state = apply_ops(initial_state, list_U)
    return measure(state)
```

But doing this by hand or using pure NumPy is tedious, so today we will shift to the quantum software framework PennyLane.

Learning outcomes

- Implement single-qubit quantum algorithms in PennyLane
- Describe the behaviour of common single-qubit gates
- Represent the state of a single qubit on the Bloch sphere

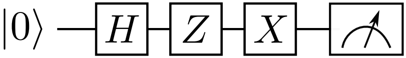
Quantum functions

Recall three of our quantum gates from last time:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

We can apply these gates to a qubit and express the computation in matrix form, or as a quantum circuit.

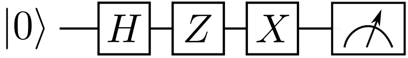
$$XZH|0\rangle$$



Quantum functions

We can also express this circuit as a **quantum function** in PennyLane.

$$XZH|0\rangle$$



```python
import pennylane as qml

def my_quantum_function():
    qml.Hadamard(wires=0)
    qml.PauliZ(wires=0)
    qml.PauliX(wires=0)
    return qml.sample()
```

Quantum functions

Quantum functions are like normal Python functions, with two special properties:

1. Apply one or more quantum operations

```
import pennylane as qml

def my_quantum_function():
    qml.Hadamard(wires=0) # Apply Hadamard gate to qubit 0
    qml.PauliZ(wires=0)   # Apply Pauli Z gate to qubit 0
    qml.PauliX(wires=0)   # Apply Pauli X gate to qubit 0
    return qml.sample()
```

Q: Why `wires`? A: PennyLane can be used for continuous-variable quantum computing, which does not use qubits.

Quantum functions

Quantum functions are like normal Python functions, with two special properties:

1. Apply one or more quantum operations
2. Return a measurement on one or more qubits

```
import pennylane as qml

def my_quantum_function():
    qml.Hadamard(wires=0)
    qml.PauliZ(wires=0)
    qml.PauliX(wires=0)
    return qml.sample() # Return measurement samples
```

Devices

Quantum functions are executed on **devices**. These can be either *simulators*, or *actual quantum hardware*.

```
import pennylane as qml

dev = qml.device('default.qubit', wires=1, shots=100)
```

This creates a device of type `'default.qubit'` with 1 qubit that returns 100 measurement samples for anything that is executed.

Quantum functions

A **QNode (quantum node)** is an object that binds a quantum function to a device, and executes it.

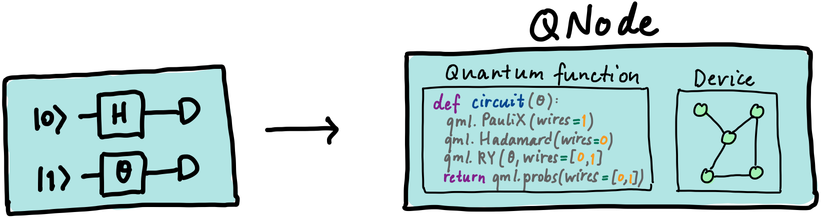


Image credit: `https://pennylane.ai/qml/glossary/quantum_node.html`

Quantum nodes

```
import pennylane as qml

dev = qml.device('default.qubit', wires=1, shots=100)

def my_quantum_function():
    qml.Hadamard(wires=0)
    qml.PauliZ(wires=0)
    qml.PauliX(wires=0)
    return qml.sample()
```

With these two components, we can create and execute a QNode.

```
# Create a QNode
my_qnode = qml.QNode(my_quantum_function, dev)

# Execute the QNode
result = my_qnode()
```

Let's go do it!

You probably have some questions...

1. Where's the state?
   - Inside the device!
2. What happens to the gates?
   - Operations are recorded onto a "tape"
   - The QNode constructs the tape when it is called
   - The tape is then executed on the device.

More quantum gates

So far, we know 3 gates that do the following:

But a general qubit state looks like

where α and β are *complex numbers* (such that $|\alpha|^2 + |\beta|^2 = 1$).

How do we make the rest?

Z rotations

Consider the operation Z:

$$Z|0\rangle = |0\rangle, \quad Z|1\rangle = -|1\rangle.$$

Apply this to a superposition:

The *sign* of the amplitude on the $|1\rangle$ state has changed.

We know that $-1 = e^{i\pi}$:

What if instead of π, we used an arbitrary angular parameter?

The extra $e^{i\theta}$ is called a **relative phase**.

The "proper" form of this rotation is

$$RZ(\theta) = e^{-i\frac{\theta}{2}Z} = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$$

In PennyLane, it is called like this:

```
qml.RZ(theta, wires=wire)
```

Exercise: expand out the exponential of Z to obtain the matrix representation.

S and T

Two other special cases: $\theta = \pi/2$, and $\theta = \pi/4$.

$$\begin{aligned} S &= RZ(\pi/2) = \begin{pmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \\ T &= RZ(\pi/4) = \begin{pmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \end{aligned}$$

In PennyLane:

```
qml.S(wires=wire)
qml.T(wires=wire)
```

S is part of a special group called the **Clifford group**.

T is used in universal gate sets for fault-tolerant QC.

Exercise: In PennyLane, implement the circuit below

Run your circuit with two different values of θ and take 1000 shots.

How does θ affect the measurement outcome probabilities?

X and Y rotations

RZ changes the phase, but not the magnitudes of the amplitudes.
How do we change those?

RX, and RY rotations...

“Rotations”?

There is a reason we are calling these rotations.

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

We can rewrite $\alpha = ae^{i\phi}$ and $\beta = be^{i\omega}$ where a, b are real-valued numbers:

Factor out the $e^{i\phi}$ (a **global phase**):

“Rotations”?

The global phase doesn’t matter though!

It does not affect the measurement outcome probabilities.

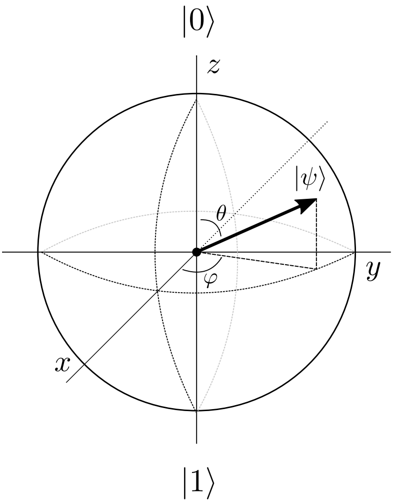
Relabel:

"Rotations"?

Normalization tells us that $a^2 + b^2 = 1$. What else has this relationship?

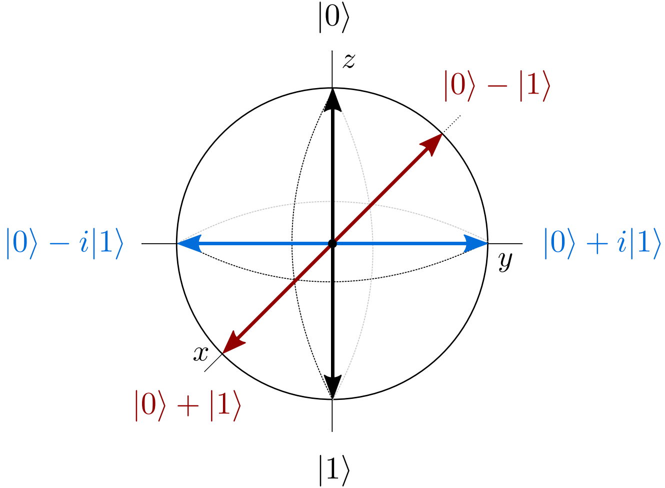
We can rewrite as:

So any single-qubit state can be specified by two angular parameters... just like points on a sphere!



`https://javafxpert.github.io/grok-bloch/`

Rotations: the Bloch sphere



Rotations: the Bloch sphere

RX, RY, and RZ correspond visually to rotations about their respective axes.

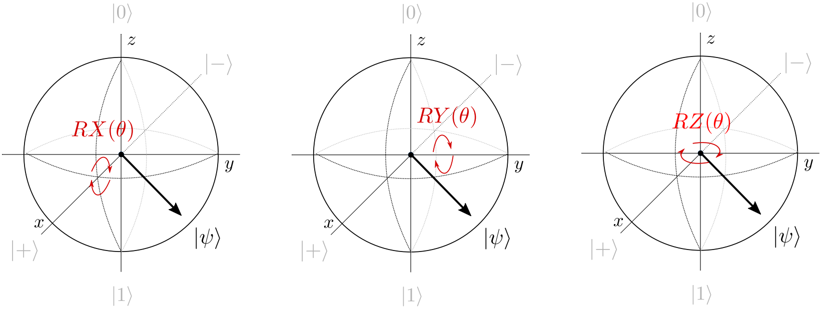
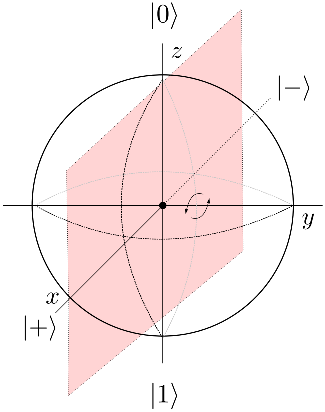
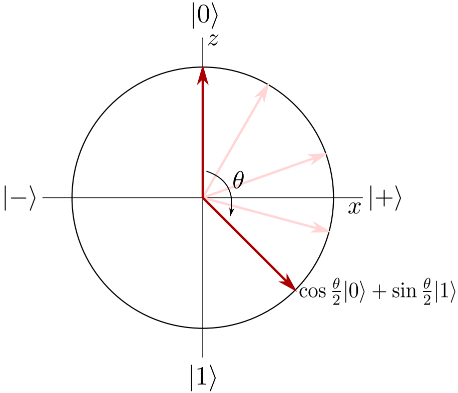


Image credit: Codebook node I.6

Rotations: RY

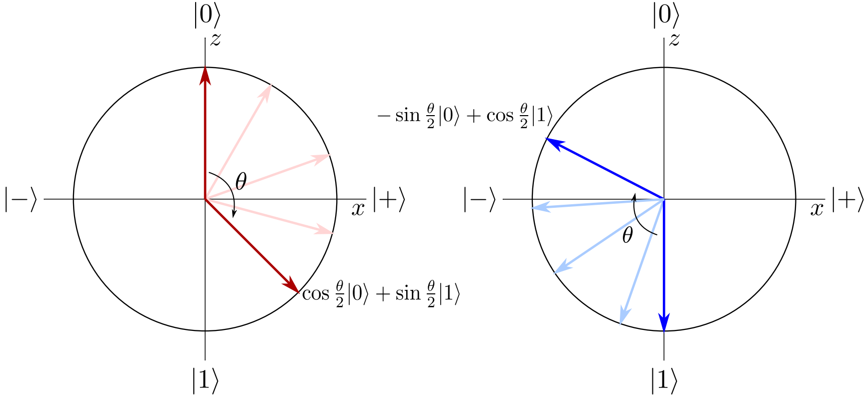




Rotations: RY

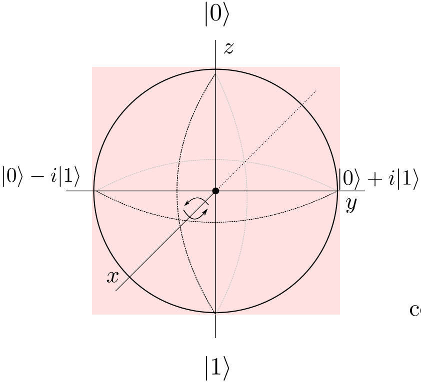
The matrix representation of RY is

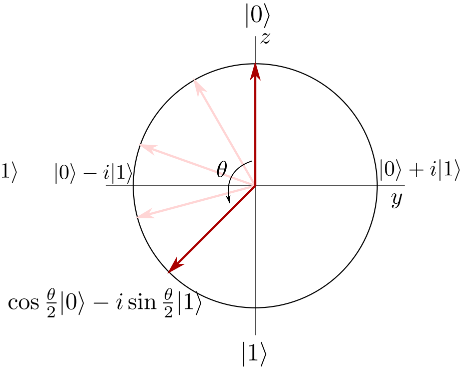
$$RY(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$



RX is similar but has complex components:

$$RX(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$





Pauli rotations

These unitary operations are called **Pauli rotations**.

	Math	Matrix	Code	Special cases
RZ	$e^{-i\frac{\theta}{2}Z}$	$\begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$	`qml.RZ`	$Z(\pi), S(\pi/2), T(\pi/4)$
RY	$e^{-i\frac{\theta}{2}Y}$	$\begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$	`qml.RY`	$Y(\pi)$
RX	$e^{-i\frac{\theta}{2}X}$	$\begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$	`qml.RX`	$X(\pi), SX(\pi/2)$

Pauli rotations

Exercise: design a quantum circuit to prepare the state

$$|\psi\rangle = \frac{\sqrt{3}}{2}|0\rangle - \frac{1}{\sqrt{2}}e^{i\frac{5}{4}}|1\rangle$$

Hint: you can also return the state or measurement outcome probabilities in PennyLane:

```
@qml.qnode(dev)
def some_circuit():
    # Gates...
    # return qml.probs(wires=0)
    return qml.state()
```

What about H?

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

This does not have the form of RX, RY, or RZ.

But, we can use a combination of these to make an H (actually, just need two of the three).

Deep dive: unitary operations

The $n \times n$ unitary matrices are a mathematical group under matrix multiplication, $U(n)$:

1. Closure: for U, V unitary, UV is also unitary
2. Associativity: $(UV)W = U(VW)$
3. Identity: $\mathbb{1}$
4. Inverses: $U^{-1} = U^\dagger$

Deep dive: unitary operations

The $n \times n$ unitary matrices are a mathematical group under matrix multiplication, $U(n)$:

1. Closure: for U, V unitary, UV is also unitary
2. Associativity: $(UV)W = U(VW)$
3. Identity: $\mathbb{1}$
4. Inverses: $U^{-1} = U^\dagger$

Any unitary matrix can be written in terms of a finite set of real-valued parameters:

$$U(\phi, \theta, \omega) = e^{i\alpha} \begin{pmatrix} e^{-i(\phi+\omega)/2} \cos(\theta/2) & -e^{i(\phi-\omega)/2} \sin(\theta/2) \\ e^{-i(\phi-\omega)/2} \sin(\theta/2) & e^{i(\phi+\omega)/2} \cos(\theta/2) \end{pmatrix}$$

Universal gate sets: Pauli rotations

With just RZ and RY (or RZ/RX, RY/RX), we can implement *any single-qubit unitary operation*[1]:

$$U = e^{i\alpha} RZ(\omega) RY(\theta) RZ(\phi)$$

$\{RZ, RY\}$ is **universal** for single-qubit quantum computing.

Hands-on...

For more fun: do text exercises in Codebook node I.3 and I.7.

[1] Note that the α technically doesn't matter.

Universal gate sets: H and T

With just H and T, we can approximate any single-qubit rotation up to arbitrary accuracy. For example, we can implement $RZ(0.1)$ up to accuracy 10^{-10}:

```
→ gridsynth 0.1 -d 10
HTHTHTHTHTSHTHTHTHTSHTSHTHTHTSHTSHTHTSHTHTSHTHTHTSHTSHTHTSHTSHTSHTS
HTHTHTHTHTHTHTHTHTHTHTSHTSHTSHTSHTSHTSHTHTSHTSHTSHTSHTHTHTSHTSHTHT
SHTSHTSHTHTHTHTSHTHTHTSHTSHTHTHTHTHTSHTHTHTSHTSHTSHTSHTSHTHTSHTHTHT
HTHTHTHTHTSHTHTHTSHTHTSHTSHTHTHTSHTSHTHTSHTSHTHXWWW
```

This was generated using the `newsynth` Haskell package:
`https://www.mathstat.dal.ca/~selinger/newsynth/`

Universal gate sets: H and T

Or to accuracy 10^{-100}:

```
→ gridsynth 0.1 -d 100
SHTSHTHTSHTSHTHTSHTHTSHTSHTHTSHTHTSHTSHTSHTHTSHTHTSHTSHTSHTHTHTSHTHTSHTHTHTHTS
HTSHTHTSHTSHTSHTSHTSHTHTHTHTSHTHTHTHTHTSHTHTSHTSHTHTHTSHTHTSHTHTHTSHTHTSHTSHTS
HTHTSHTHTHTSHTSHTSHTSHTSHTSHTHTHTSHTSHTSHTHTHTSHTHTHTHTHTSHTHTSHTHTSHTHTSHTSHT
SHTSHTHTSHTHTSHTSHTHTHTSHTHTHTSHTSHTHTHTHTHTHTHTHTHTHTHTHTHTHTSHTSHTSHTSHTHTHT
SHTSHTSHTHTSHTHTHTSHTHTSHTSHTSHTHTHTHTHTHTHTSHTSHTSHTSHTHTHTHTSHTSHTSHTSHTSHTH
THTHTSHTSHTSHTSHTSHTHTHTHTHTSHTHTHTSHTHTSHTHTHTSHTHTHTSHTHTSHTHTSHTSHTSHTSHTHT
HTHTSHTSHTSHTHTSHTHTSHTSHTHTHTHTHTHTHTSHTHTHTHTHTSHTHTHTHTSHTSHTSHTSHTSHTSHTSH
TSHTSHTSHTSHTSHTSHTSHTHTSHTHTSHTSHTSHTHTSHTSHTSHTSHTHTHTSHTHTHTHTHTHTHTHTHTHTS
HTHTHTHTHTSHTSHTHTHTSHTHTSHTSHTSHTSHTHTHTSHTSHTSHTHTHTSHTHTHTHTHTSHTSHTSHTSHTS
HTHTSHTSHTHTHTHTHTHTHTHTSHTSHTHTSHTSHTHTHTSHTSHTHTSHTSHTSHTSHTSHTHTHTSHTHTS
HTSHTHTHTHTHTSHTSHTSHTSHTSHTSHTSHTSHTHTSHTSHTHTHTSHTSHTHTHTSHTSHTSHTHTSHTHTSHT
HTHTHTSHTHTSHTHTSHTSHTHTSHTHTSHTHTHTSHTSHTHTSHTHTHTHTHTHTHTSHTHTSHTHTHTSHTHTSH
TSHTHTSHTHTHTSHTHTSHTSHTHTSHTSHTHTSHTHTSHTSHTSHTSHTHTSHTHTSHTHTSHTSHTSHTSHTHTH
THTHTSHTHTSHTHTSHTHTSHTHTSHTSHTHTHTHTSHTSHTHTHTHTSHTHTSHTSHTHTHTSHTHTSHTSHTSHT
SHTHTSHTSHTSHTSHTHTHTSHTHTHTHTSHTHTHTSHTHTSHTHTHTSHTSHTSHTSHTSHTSHTSHTHTHTHTSH
TSHTSHTSHTHTSHTSHTSHTHTHTSHTSHTHTSHTHTSHTSHTSHTHTSHTHTHTHTSHTSHTHTHTHTHTSHTHTH
TSHTHTSHTSHTSHTHTSHTHTSHTHTHTHTSHTHTSHTHTSHTHTSHTHTSHTHTHTHTSHTSHTSHTHTHTSHTHT
SHTHTHTSHTSHTSHTSHTHTSHTSHTHTHTHTSHTSHTHTSHTSHTSHTSHTHTHTSHTHTHTSHTSHTHTHTSHTH
TSHTHTHTSHTSHTHTHTSHTHTSHTHTHTHTHTSHTSHTSHTHTSHTSHTSHTSHTSHTSHTSHTSHTHTHTHTSHT
SHTHTHTSHTSHTSHTSHTSHTSHTSHTSHTSHTSHTHTSHTHTSHTHTHTSHTHTSHTSHTSHTHTHTSHTSHTHTH
THTSHTHTSHTHTHTSHTSHTHTSHTHTSHTSHTSHTHTHTHTSHTSHTHTHTHTSHTHTSHTSHTSHTHTHTHTHTH
THTSHTSHTHTHTHTHTSHTSHTHTHTSHTSHTSHTHTHTSHTHTHTSHTHTSHTSHTHTHTSHTSHTHTSHTSHTSH
THTHTSHTHTHTHTHTSHTHTHTSHTSHTHTHTSHTHTHTSHTSHTSHTHTSHTSHTHTHTSHTSHTHTSHTSHTHTH
TSHTHTSHTSHTHTSHTHTSHTSHTHTSHTHTSHTHTHTSHTSHTSHTSHTSHTSHTSHTHTSHTHTSHTSHTHTHTS
HTHTSHTHTSHTHTHTSHTSHTSHTHTHTSHTHTSHTHTSHTHTSHTSHTSHTSHTSHTHTHTSHTSHTSHTSHTHTS
HTHTHTHTHTHTHTHTSHTHTHTHTSHTHTHTHTHTSHTSHTSHTHTHTSHTHTSHTHTSHTSHTSHTHTSHTHTSHT
HTHTHTSHTSHTSHTHTSHTSHTHTSHTSHTSHTSHTHTHTSHTSHTSHTHTHTSHTHTSHTSHTHTSHTSHTHTSHT
SHTSHTSHTHTHTSHTHTSHTHTSHTSHTHTSHTHTHTHTSHTSHTSHTSHTSHTSHTSHTHTSHTSHTHTHTSHTSH
THTHTHTHTHTHTHTSHTSHTSHTSHTHTSHTSHTSHTHTHTSHTSHTSHTHTSHTHTSHTSHTHTHTHTSHTSHTSH
THTHTSHTHTSHTSHTSHTHTHTSHTSHTSHTHTHTHTHTHTHTHTSHTHTSHTSHTHTHTHTSHTSHTSHTSHTHTS
HTHTHTHTHTHTHTSHTSHTHTSHTHTHTSHTSHTSHTHTHTHTSHTHTHTHTHTHTSHTHTHTHTHTSHTHTSHTSH
TSHTSHTHTSHTHTSHTSHTSHTSHTHTSHTHTSHTHTSHTHTHTHTSHTSHTSHTSHTHTSHTHTSHTSHTSHTSHT
HTHTHTHTHTSHTSHTSHTHTHS
```

...we'll talk more about this in a few weeks when we discuss *quantum compilation*.

Recap

- Implement single-qubit quantum algorithms in PennyLane
- Describe the behaviour of common single-qubit gates
- Represent the state of a single qubit on the Bloch sphere

Next time

Content:

- The theory of measurements
- Expectation values
- Measuring in different bases

Action items:

1. Finish Assignment 0 (due Monday evening)
2. Quiz next class

Recommended reading:

- Codebook nodes I.1-I.10
- Nielsen & Chuang 4.2