Codercise I.4.1 — Flipping bits

Open related theory

A common use of the X gate is in initializing the state of a qubit at the beginning of an algorithm. Quite often, we would like our qubits to start in state $|0\rangle$ (which is the default in PennyLane), however there are many cases where we instead would like to start from $|1\rangle$. Complete the function below by using `qml.PauliX` to initialize the qubit's state to $|0\rangle$ or $|1\rangle$ based on an input flag. Then, use `qml.QubitUnitary` to apply the provided unitary `U`.

Solution :

```
dev = qml.device("default.qubit", wires=1)

U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)

@qml.qnode(dev)
def varied_initial_state(state):
    """Complete the function such that we can apply the operation U to
    either |0> or |1> depending on the input argument flag.
    Args:
        state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,
            otherwise, leave it in state 0.
    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    ##################
    if state > 0:
        qml.PauliX(wires=0)
    ##################
    # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON THE state PARAMETER
    ##################
    # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON THE state PARAMETER
    # APPLY U TO THE STATE
    qml.QubitUnitary(U, wires=0)
    return qml.state()
```

```
    # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON THE state PARAMETER

    ##################

    # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON THE state PARAMETER

    # APPLY U TO THE STATE
    qml.QubitUnitary(U, wires=0)

    return qml.state()

```

Reset Code Submit

Correct!

Qiskit Program:

```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

dev = qml.device("default.qubit", wires=1)

U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)

@qml.qnode(dev)
def varied_initial_state(state):
    """Complete the function such that we can apply the operation U to
    either |0> or |1> depending on the input argument flag.

    Args:
        state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,
            otherwise, leave it in state 0.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    ##################
    if state > 0:
        qml.PauliX(wires=0)
    ##################

    # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON THE state PARAMETER

    # APPLY U TO THE STATE
    qml.QubitUnitary(U, wires=0)

    return qml.state()
state = 0  # can be 1 or 0
circuit = qml.QNode(varied_initial_state, dev)
print(circuit(state))
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

dev = qml.device("default.qubit", wires=1)

U = np.array([[1, 1], [1, -1]]) / np.sqrt(2)

@qml.qnode(dev)
def apply_u():

    ##################
    qml.QubitUnitary(U, wires=0)
    ##################

    # USE QubitUnitary TO APPLY U TO THE QUBIT

    # Return the state
    return qml.state()
```

Codercise I.4.2 — Uniform superposition

Open related theory

You might have noticed that we've been using this operation a lot:

$$U = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2}$$

This is none other than **Hadamard gate**, and is typically denoted by H. In PennyLane, it is implemented as `qml.Hadamard`.

The Hadamard gate is special because it can create a *uniform superposition* of the two states $|0\rangle$ and $|1\rangle$. Many quantum algorithms rely on us being able to create uniform superpositions, so you'll see the Hadamard gate everywhere!

Complete the quantum function below such that it:

- applies a Hadamard gate to the qubit,
- returns the *state* of the qubit with `qml.state`.

Solution:

```
dev = qml.device("default.qubit", wires=1)
@qml.qnode(dev)
def apply_hadamard():
    ##################
    qml.Hadamard(wires=0)
    ##################
    # APPLY THE HADAMARD GATE
    # RETURN THE STATE
    return qml.state()
```

```
@qml.qnode(dev)
def apply_hadamard():
    #################
    qml.Hadamard(wires=0)
    #################

    # APPLY THE HADAMARD GATE

    # RETURN THE STATE
    return qml.state()

```

Reset Code Submit

Correct!

Qiskit Program:

```
import numpy as np
import random
from qiskit.quantum_info import Statevector
```

```
import pennylane as qml
import matplotlib.pyplot as plt

dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def apply_hadamard():
    ##################
    qml.Hadamard(wires=0)
    ##################

    # APPLY THE HADAMARD GATE

    # RETURN THE STATE
    return qml.state()

circuit = qml.QNode(apply_hadamard, dev)
print(circuit())
```

O/P:

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

Codercise I.4.3 — Combining X and H

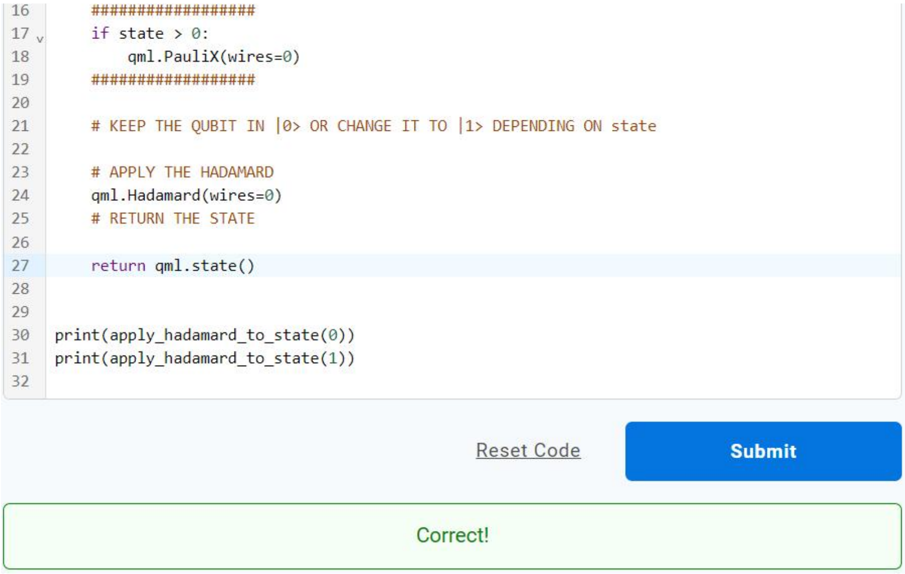
Open related theory

Combining your code from codercises I.4.1, and I.4.2, apply the Hadamard gate to both $|0\rangle$ and $|1\rangle$. What do the two different output states look like? Do you notice anything special about them?

Solution:

```
dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def apply_hadamard_to_state(state):
    """Complete the function such that we can apply the Hadamard to
    either |0> or |1> depending on the input argument flag.
    Args:
        state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,
            otherwise, leave it in state 0.
    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    ##################
    if state > 0:
        qml.PauliX(wires=0)
    ##################
    # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON state
    # APPLY THE HADAMARD
    qml.Hadamard(wires=0)
    # RETURN THE STATE
```

```
    return qml.state()

print(apply_hadamard_to_state(0))
print(apply_hadamard_to_state(1))
```



Qiskit Program:

```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def apply_hadamard_to_state(state):
    """Complete the function such that we can apply the Hadamard to
    either |0> or |1> depending on the input argument flag.

    Args:
        state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,
            otherwise, leave it in state 0.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    ##################
    if state > 0:
        qml.PauliX(wires=0)
    ##################

    # KEEP THE QUBIT IN |0> OR CHANGE IT TO |1> DEPENDING ON state

    # APPLY THE HADAMARD
    qml.Hadamard(wires=0)

    # RETURN THE STATE
```

```
    return qml.state()

print(apply_hadamard_to_state(0))
print(apply_hadamard_to_state(1))
```

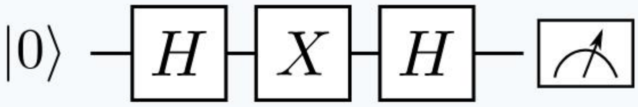
O/P:

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

Codercise I.4.4 — A QNode with X and H

Open related theory

Now let's combine what we've just learned. Create a device with one qubit. Then, write a QNode (from scratch!) that applies the following circuit and returns the state.



Determine its effect on the two basis states. What do you think this operation does?

The signature of your function should be:

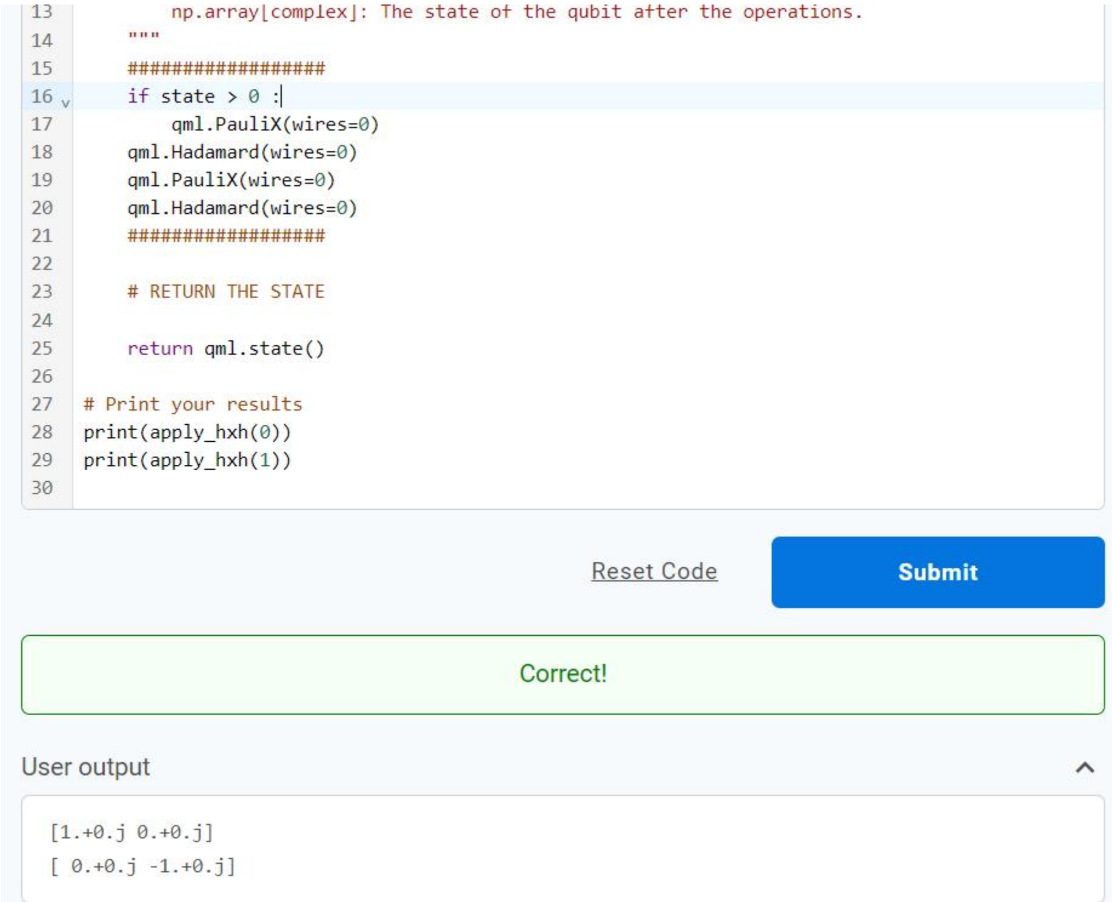
```
def apply_hxh(state):
    ...
    return qml.state()
```

where, as in the previous exercises, `state` is an integer that indicates which basis state to prepare.

Solution:

```
dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def apply_hxh(state):
    """Complete the function such that we can apply the Hadamard to
    either |0> or |1> depending on the input argument flag.

    Args:
        state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,
            otherwise, leave it in state 0.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    ##################
    if state > 0 :
        qml.PauliX(wires=0)
    qml.Hadamard(wires=0)
```

```
    qml.PauliX(wires=0)
    qml.Hadamard(wires=0)
    ##################

    # RETURN THE STATE

    return qml.state()

# Print your results
print(apply_hxh(0))
print(apply_hxh(1))
```



Qiskit Program:

```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

dev = qml.device("default.qubit", wires=1)

@qml.qnode(dev)
def apply_hxh(state):
    """Complete the function such that we can apply the Hadamard to
    either |0> or |1> depending on the input argument flag.

    Args:
        state (int): Either 0 or 1. If 1, prepare the qubit in state |1>,
```

```
        otherwise, leave it in state 0.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """
    ##################
    if state > 0 :
        qml.PauliX(wires=0)

    qml.Hadamard(wires=0)
    qml.PauliX(wires=0)
    qml.Hadamard(wires=0)

    ##################

    # RETURN THE STATE

    return qml.state()

# Print your results
print(apply_hxh(0))
print(apply_hxh(1))
```

O/P:

```
[1.+0.j 0.+0.j]
[ 0.+0.j -1.+0.j]
```

```
circuit = qml.QNode(apply_hxh, dev)
qml.drawer.use_style("pennylane")
result = qml.draw_mpl(circuit)(1)
plt.show()
```