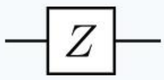
## Codercise I.5.1 — The Pauli Z gate

Open related theory

The Pauli $Z$ gate is defined by its action on the computational basis states

$$\begin{aligned} |0\rangle &\rightarrow |0\rangle, \\ |1\rangle &\rightarrow -|1\rangle. \end{aligned} \tag{1}$$

In PennyLane, you can implement it by calling `qml.PauliZ`. It has the following circuit element:



Write a QNode that applies `qml.PauliZ` to the $|+\rangle$ state and returns the state. What state is this? How do the measurement probabilities differ from those of the state $|+\rangle$?

**Solution** :

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

@qml.qnode(dev)
def apply_z_to_plus():
    """Write a circuit that applies PauliZ to the |+> state and returns
    the state.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """

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

    # CREATE THE |+> STATE

    # APPLY PAULI Z
    qml.PauliZ(wires=0)
    # RETURN THE STATE
    return qml.state()

print(apply_z_to_plus())
```

```
@qml.qnode(dev)
def apply_z_to_plus():
    """Write a circuit that applies PauliZ to the |+> state and returns
    the state.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """

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

    # CREATE THE |+> STATE

    # APPLY PAULI Z
    qml.PauliZ(wires=0)
    # RETURN THE STATE
    return qml.state()


print(apply_z_to_plus())
```

Reset Code Submit

Correct!

**Qiskit Program**:

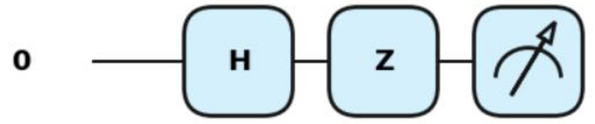
```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

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

@qml.qnode(dev)
def apply_z_to_plus():
    """Write a circuit that applies PauliZ to the |+> state and returns
    the state.

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

    # CREATE THE |+> STATE

    # APPLY PAULI Z
    qml.PauliZ(wires=0)

    # RETURN THE STATE
```

```
    return qml.state()

print(apply_z_to_plus())
```

**<u>O/P</u>**:

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

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




## Codercise I.5.2 — The Z Rotation

Open related theory

Given some arbitrary $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and angle of rotation $\omega$ (in radians), the $Z$ rotation gate $RZ$ acts as follows.

$$RZ(\omega)|\psi\rangle = e^{-i\frac{\omega}{2}}\alpha|0\rangle + \beta e^{i\frac{\omega}{2}}|1\rangle. \tag{3}$$

However, this prefactor of $e^{-i\frac{\omega}{2}}$ is also a **global phase**, and can thus be factored out. This means that $RZ(\omega)$ produces

$$RZ(\omega)|\psi\rangle = e^{-i\frac{\omega}{2}}\alpha|0\rangle + \beta e^{i\frac{\omega}{2}}|1\rangle \sim \alpha|0\rangle + \beta e^{i\omega}|1\rangle. \tag{4}$$

In PennyLane, this operation is accessible as `qml.RZ`, which is a parametrized operation, and so we must specify not only a wire, but an angle of rotation:

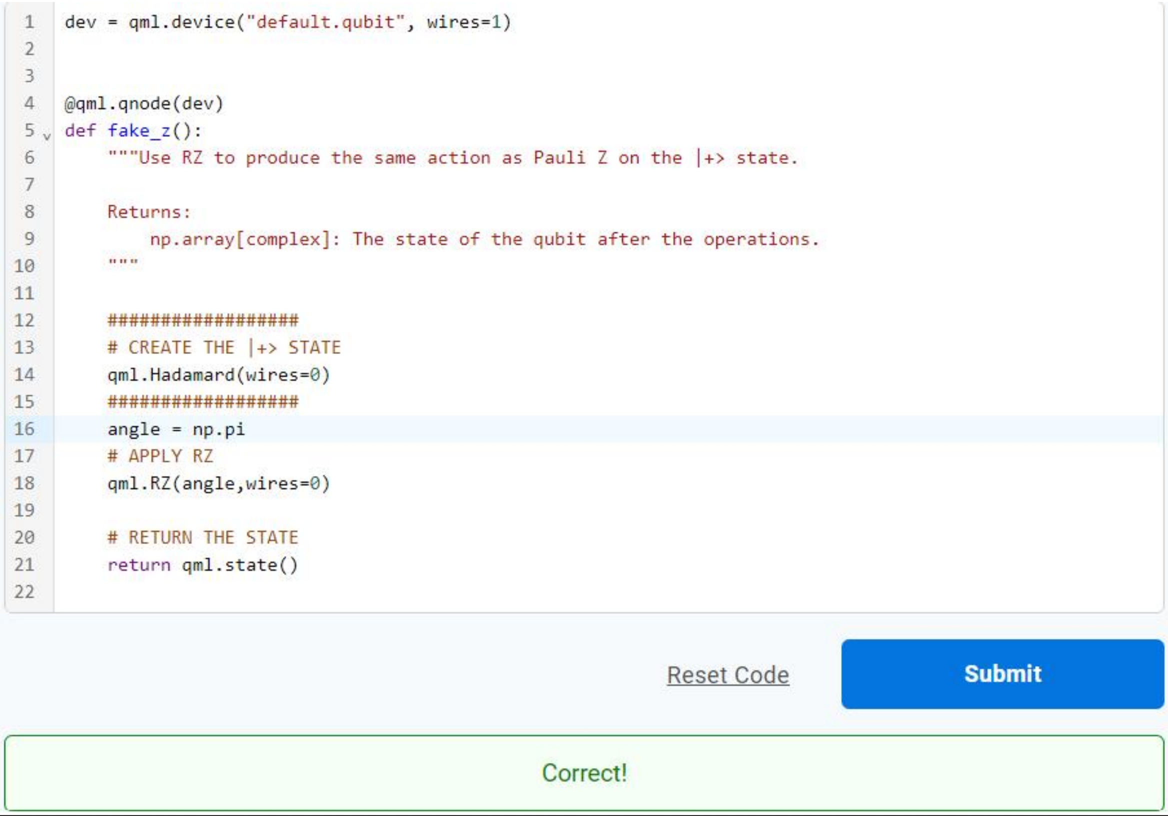
```
qml.RZ(angle, wires=wire)
```

Write a QNode that uses `qml.RZ` to simulate a `qml.PauliZ` operation and return the state. Apply it to the $|+\rangle$ state to check your work.

**Solution:**

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

@qml.qnode(dev)
def fake_z():
```

```
    """Use RZ to produce the same action as Pauli Z on the |+> state.
    Returns:
      np.array[complex]: The state of the qubit after the operations.
    """
    ##################
    # CREATE THE |+> STATE
    qml.Hadamard(wires=0)
    ##################
    angle = np.pi
    # APPLY RZ
    qml.RZ(angle,wires=0)

    # RETURN THE STATE
    return qml.state()
```




**Qiskit Program**:

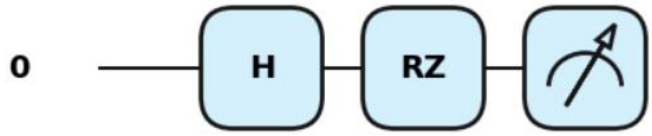
```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

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

@qml.qnode(dev)
def fake_z():
    """Use RZ to produce the same action as Pauli Z on the |+> state.

    Returns:
      np.array[complex]: The state of the qubit after the operations.
    """
```

```
    ##################
    # CREATE THE |+> STATE
    qml.Hadamard(wires=0)
    ##################

    # APPLY RZ
    qml.RZ(np.pi,wires=0)

    # RETURN THE STATE
    return qml.state()

print(fake_z())
```

**O/P**:

```
[4.32978028e-17-0.70710678j 4.32978028e-17+0.70710678j]
```

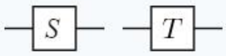
```
circuit = qml.QNode(fake_z, dev)
qml.drawer.use_style("pennylane")
result = qml.draw_mpl(circuit)()
plt.show()
```



## Codercise I.5.3 — The S and T gates

Open related theory

The quarter turn $RZ(\pi/2)$ and eighth turn $RZ(\pi/4)$ gates also have their own names: the **phase gate**, $S$, and the $T$ **gate**, respectively.



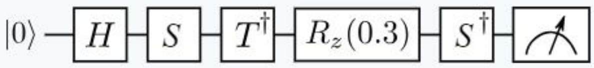
In PennyLane, they are implemented directly as the non-parametrized operations `qml.S` and `qml.T`.

Adjoints in PennyLane can be computed by applying the `qml.adjoint` transform to an operation before specifying its parameters and wires. For example,

```
qml.adjoint(qml.RZ)(omega, wires=0)
```

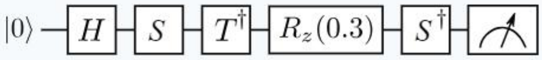
performs the same computation as `qml.RZ(-omega, wires=0)`, since $RZ^{\dagger}(\omega) = RZ(-\omega)$.

With the above in mind, implement the circuit below, using adjoints when necessary, and return the quantum state.



**Solution:**

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

@qml.qnode(dev)
def many_rotations():
    """Implement the circuit depicted above and return the quantum state.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """

    ##################
    # CREATE THE |+> STATE
    qml.Hadamard(wires=0)
    ##################

    # IMPLEMENT THE CIRCUIT
    qml.S(wires=0)
    qml.adjoint(qml.T)(wires=0)
    qml.RZ(0.3,wires=0)
    qml.adjoint(qml.S)(wires=0)
    # RETURN THE STATE

    return qml.state()
```




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


@qml.qnode(dev)
def many_rotations():
    """Implement the circuit depicted above and return the quantum state.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """

    ##################
    # CREATE THE |+> STATE
    qml.Hadamard(wires=0)
    ##################

    # IMPLEMENT THE CIRCUIT
    qml.S(wires=0)
    qml.adjoint(qml.T)(wires=0)
    qml.RZ(0.3,wires=0)
    qml.adjoint(qml.S)(wires=0)
    # 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 many_rotations():
    """Implement the circuit depicted above and return the quantum state.

    Returns:
        np.array[complex]: The state of the qubit after the operations.
    """

    ##################
    # CREATE THE |+> STATE
    qml.Hadamard(wires=0)
    ##################

    # IMPLEMENT THE CIRCUIT
    qml.S(wires=0)
    qml.adjoint(qml.T)(wires=0)
    qml.RZ(0.3,wires=0)
    qml.adjoint(qml.S)(wires=0)
    # RETURN THE STATE
    return qml.state()
```

print(many_rotations())

O/P:

```
[0.69916673-0.10566872j 0.56910461-0.41966647j]
```

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