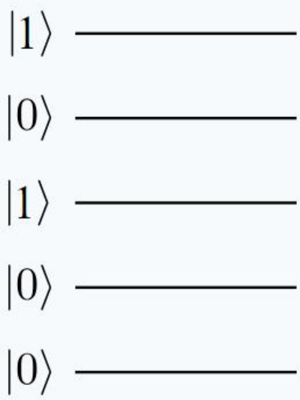
Important: qubit-ordering convention. In PennyLane, qubits are indexed numerically from left to right. Therefore, a state such as $|10100\rangle$ indicates that the first and third qubit (or, wires `0` and `2`) are in state $|1\rangle$, and the second, fourth, and fifth qubit are in state $|0\rangle$. When drawing quantum circuits, our convention is that the leftmost (first) qubit is at the *top* of the circuit, such that qubits starting in state $|10100\rangle$ correspond to the circuit below:



A different convention, where qubit `0` is the rightmost qubit in the ket, is used in a number of other quantum computing software frameworks and resources. Always check the qubit ordering when you start using a new software library!

For this codercise, you will write a circuit in PennyLane that accepts an integer value, then prepares and returns the corresponding computational basis state vector $|n\rangle$. (Assume a 3-qubit device). Try a few examples; does the appearance of the state vector match what you expect given the integer?

Solution :

▼ *Hint.*

You will find the `numpy` function `np.binary_repr` helpful for this challenge.

▼ *Hint.*

There are two ways to solve this challenge. The first is to manipulate the individual qubits based on the bit values. The second is to use a built-in state preparation template. Check out the PennyLane template library and see if there are any predefined functions that will help you.

https://docs.pennylane.ai/en/stable/code/api/pennylane.BasisStatePreparation.html
https://numpy.org/doc/stable/reference/generated/numpy.binary_repr.html

```
num_wires = 3
dev = qml.device("default.qubit", wires=num_wires)


@qml.qnode(dev)
def make_basis_state(basis_id):
    """Produce the 3-qubit basis state corresponding to |basis_id>.

    Note that the system starts in |000>.

    Args:
        basis_id (int): An integer value identifying the basis state to construct.
```

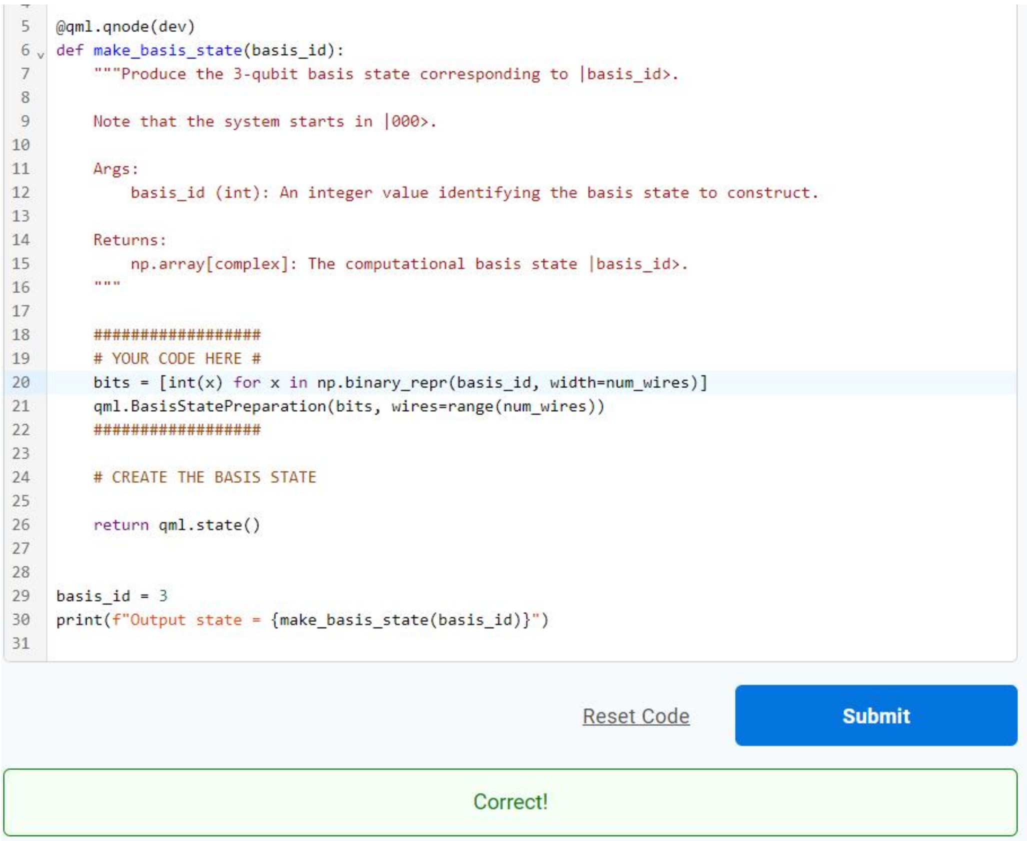
```
    Returns:
        np.array[complex]: The computational basis state |basis_id>.
    """

    ##################
    # YOUR CODE HERE #
    bits = [int(x) for x in np.binary_repr(basis_id, width=num_wires)]
    qml.BasisStatePreparation(bits, wires=range(num_wires))
    ##################

    # CREATE THE BASIS STATE

    return qml.state()

basis_id = 3
print(f"Output state = {make_basis_state(basis_id)}")
```



```
@qml.qnode(dev)
def make_basis_state(basis_id):
    """Produce the 3-qubit basis state corresponding to |basis_id>.

    Note that the system starts in |000>.

    Args:
        basis_id (int): An integer value identifying the basis state to construct.

    Returns:
        np.array[complex]: The computational basis state |basis_id>.
    """

    ##################
    # YOUR CODE HERE #
    bits = [int(x) for x in np.binary_repr(basis_id, width=num_wires)]
    qml.BasisStatePreparation(bits, wires=range(num_wires))
    ##################

    # CREATE THE BASIS STATE

    return qml.state()


basis_id = 3
print(f"Output state = {make_basis_state(basis_id)}")
```


Qiskit Program:

```
import numpy as np
import random
import pennylane as qml
import matplotlib.pyplot as plt

num_wires = 3
dev = qml.device("default.qubit", wires=num_wires)

@qml.qnode(dev)
def make_basis_state(basis_id):
```

```
    """Produce the 3-qubit basis state corresponding to |basis_id>.

    Note that the system starts in |000>.

    Args:
        basis_id (int): An integer value identifying the basis state to construct.

    Returns:
        np.array[complex]: The computational basis state |basis_id>.
    """

    ##################
    # YOUR CODE HERE #
    # Prepare the basis state |basis_id>
    #Option 1:
    #bits = [int(x) for x in np.binary_repr(basis_id, width=num_wires)]
    #qml.BasisStatePreparation(bits, wires=[0, 1, 2])
    #Option 2:
    bits = [int(x) for x in np.binary_repr(basis_id, width=num_wires)]
    qml.BasisStatePreparation(bits, wires=range(num_wires))
    ##################

    # CREATE THE BASIS STATE

    return qml.state()

basis_id = 3
print(f"Output state = {make_basis_state(basis_id)}")
```

O/P:

```
Output state = [0.+0.j 0.+0.j 0.+0.j 1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
```

Codercise I.11.2 — Separable Operations

Open related theory

Use PennyLane to create the state $|+1\rangle = |+\rangle \otimes |1\rangle$. Then, return two measurements:

- the expectation value of Y on the first qubit
- the expectation value of Z on the second qubit

In PennyLane, you can return measurements of multiple observables as a tuple, as long as they don't share wires.

Solution:

```
# Creates a device with *two* qubits
dev = qml.device("default.qubit", wires=2)


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

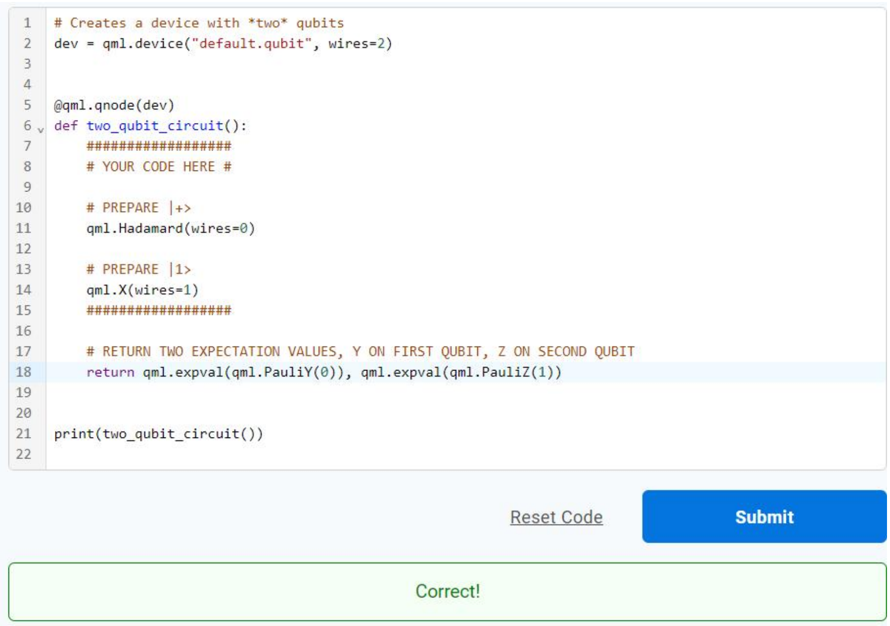
```
    #################
    # YOUR CODE HERE #

    # PREPARE |+>
    qml.Hadamard(wires=0)

    # PREPARE |1>
    qml.X(wires=1)
    #################

    # RETURN TWO EXPECTATION VALUES, Y ON FIRST QUBIT, Z ON SECOND QUBIT
    return qml.expval(qml.PauliY(0)), qml.expval(qml.PauliZ(1))

print(two_qubit_circuit())
```



```
# Creates a device with *two* qubits
dev = qml.device("default.qubit", wires=2)


@qml.qnode(dev)
def two_qubit_circuit():
    #################
    # YOUR CODE HERE #

    # PREPARE |+>
    qml.Hadamard(wires=0)

    # PREPARE |1>
    qml.X(wires=1)
    #################

    # RETURN TWO EXPECTATION VALUES, Y ON FIRST QUBIT, Z ON SECOND QUBIT
    return qml.expval(qml.PauliY(0)), qml.expval(qml.PauliZ(1))


print(two_qubit_circuit())
```


Qiskit Program:

```
import numpy as np
import random
import pennylane as qml
import matplotlib.pyplot as plt

# Creates a device with *two* qubits
dev = qml.device("default.qubit", wires=2)


@qml.qnode(dev)
def two_qubit_circuit():
    #################
    # YOUR CODE HERE #

    # PREPARE |+>
    qml.Hadamard(wires=0)

    # PREPARE |1>
    qml.X(wires=1)
```

```
    ##################

    # RETURN TWO EXPECTATION VALUES, Y ON FIRST QUBIT, Z ON SECOND QUBIT
    #return qml.probs(wires=[0, 1])
    return qml.expval(qml.PauliY(0)), qml.expval(qml.PauliZ(1))


print(two_qubit_circuit())
```

O/P:

```
(tensor(0., requires_grad=True), tensor(-1., requires_grad=True))
```

Codercise I.11.3 — Expectation value of two-qubit observable

Open related theory

Write a PennyLane circuit that creates the state $|1-\rangle = |1\rangle \otimes |-\rangle$. Then, measure the expectation value of the *two-qubit observable* $Z \otimes X$. In PennyLane, you can combine observables using the `@` symbol to represent the tensor product, e.g., `qml.PauliZ(0) @ qml.PauliZ(1)`.

Solution:

```
dev = qml.device("default.qubit", wires=2)


@qml.qnode(dev)
def create_one_minus():
    ##################
    # YOUR CODE HERE #
    ##################

    # PREPARE |1>|->
    # PREPARE |1>
    qml.X(wires=0)
    # PREPARE |->
    qml.X(wires=1)
    qml.Hadamard(wires=1)

    # RETURN A SINGLE EXPECTATION VALUE Z \otimes X
    op = qml.PauliZ(0) @ qml.PauliX(1)
    return qml.expval(op)

print(create_one_minus())
```

```python
dev = qml.device("default.qubit", wires=2)


@qml.qnode(dev)
def create_one_minus():
    ##################
     # YOUR CODE HERE #
     ##################

     # PREPARE |1>|->
     # PREPARE |1>
     qml.X(wires=0)
     # PREPARE |->
     qml.X(wires=1)
     qml.Hadamard(wires=1)

     # RETURN A SINGLE EXPECTATION VALUE Z \otimes X
     op = qml.PauliZ(0) @ qml.PauliX(1)
     return qml.expval(op)


print(create_one_minus())
```

Reset Code   Submit

Correct!

Qiskit Program:

```python
import numpy as np
import random
import pennylane as qml
import matplotlib.pyplot as plt

# Creates a device with *two* qubits
dev = qml.device("default.qubit", wires=2)


@qml.qnode(dev)
def create_one_minus():
    ##################
    # YOUR CODE HERE #
    ##################

    # PREPARE |1>|->
    # PREPARE |1>
    qml.X(wires=0)
    # PREPARE |->
    qml.X(wires=1)
    qml.Hadamard(wires=1)

    # RETURN A SINGLE EXPECTATION VALUE Z \otimes X
    op = qml.PauliZ(0) @ qml.PauliX(1)
    return qml.expval(op)

print(create_one_minus())
```

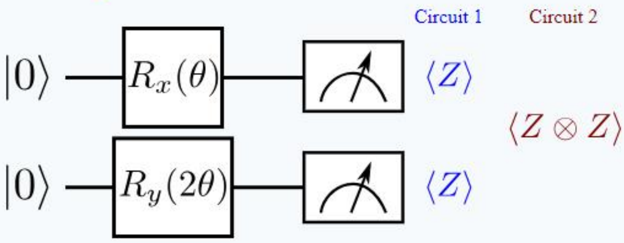
O/P:

```
0.9999999999999996
```

Codercise I.11.4 — Double Trouble

Open related theory

Implement the following circuit twice. For one version, measure the observables Z on the first qubit (i.e., $Z \otimes I$), and Z on the second qubit ($I \otimes Z$). For the other version, measure the observable $Z \otimes Z$. How do think the results of the first circuit will relate to those of the second? Plot the results as a function of θ to test your hypothesis.



Tip. In PennyLane, you don't need to specify the identity portion of observables. For example, $I \otimes Z$ is simply `qml.PauliZ(1)` rather than `qml.Identity(0) @ qml.PauliZ(1)`.

Refer : https://discuss.pennylane.ai/t/any-thought-on-i11-4/1510/4

Solution:

Qiskit Program: