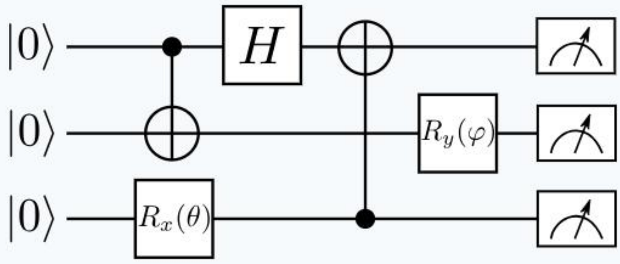
Codercise I.2.1 — Order of operations

Open related theory

The code below is a quantum function with all the gates from the circuit shown below. However, the gates are out of order! Re-arrange the lines of the function to match the order of operations in the circuit.



Solution :

```
def my_circuit(theta, phi):
    ##################
    qml.CNOT(wires=[0, 1])
    qml.Hadamard(wires=0)
    qml.RX(theta, wires=2)
    qml.CNOT(wires=[2, 0])
    qml.RY(phi, wires=1)
    ##################

    # REORDER THESE 5 GATES TO MATCH THE CIRCUIT IN THE PICTURE

    # This is the measurement; we return the probabilities of all possible output states
    # You'll learn more about what types of measurements are available in a later node
    return qml.probs(wires=[0, 1, 2])
```

```
def my_circuit(theta, phi):
    ##################
    qml.CNOT(wires=[0, 1])
    qml.Hadamard(wires=0)
    qml.RX(theta, wires=2)
    qml.CNOT(wires=[2, 0])
    qml.RY(phi, wires=1)
    ##################

    # REORDER THESE 5 GATES TO MATCH THE CIRCUIT IN THE PICTURE

    # This is the measurement; we return the probabilities of all possible output states
    # You'll learn more about what types of measurements are available in a later node
    return qml.probs(wires=[0, 1, 2])

```

Reset Code
Submit

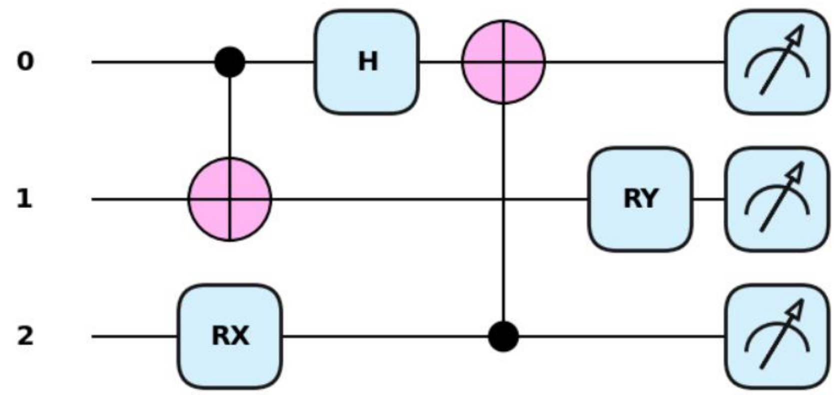
Correct!

Qiskit Program:

```
import numpy as np
import random
```

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

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

@qml.qnode(dev)
def my_circuit(theta, phi):
    ##################
    qml.CNOT(wires=[0, 1])
    qml.Hadamard(wires=0)
    qml.RX(theta, wires=2)
    qml.CNOT(wires=[2, 0])
    qml.RY(phi, wires=1)
    ##################

    # REORDER THESE 5 GATES TO MATCH THE CIRCUIT IN THE PICTURE
    # This is the measurement; we return the probabilities of all possible output states
    # You'll learn more about what types of measurements are available in a later node
    return qml.probs(wires=[0, 1, 2])

theta = np.pi/2
phi = np.pi

circuit = qml.QNode(my_circuit, dev_unique_wires)
qml.drawer.use_style("pennylane")
probs = qml.draw_mpl(circuit)(theta, phi)
plt.show()
```

O/P:



Codercise I.2.2 — Building a QNode

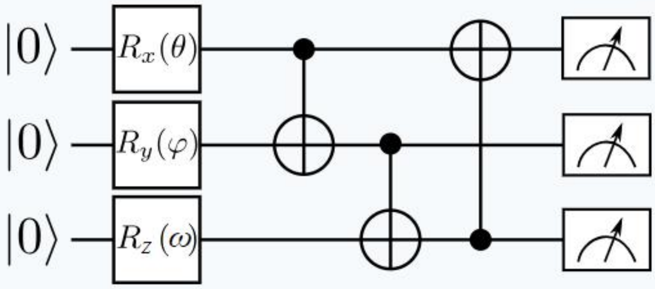
Open related theory

Recall that one way in which we can turn our quantum circuits into QNodes is via the `qml.QNode` function:

```
my_qnode = qml.QNode(my_circuit, my_device)
```

Once a QNode is created, it can be called like a function using the same parameters as the quantum function upon which it's built.

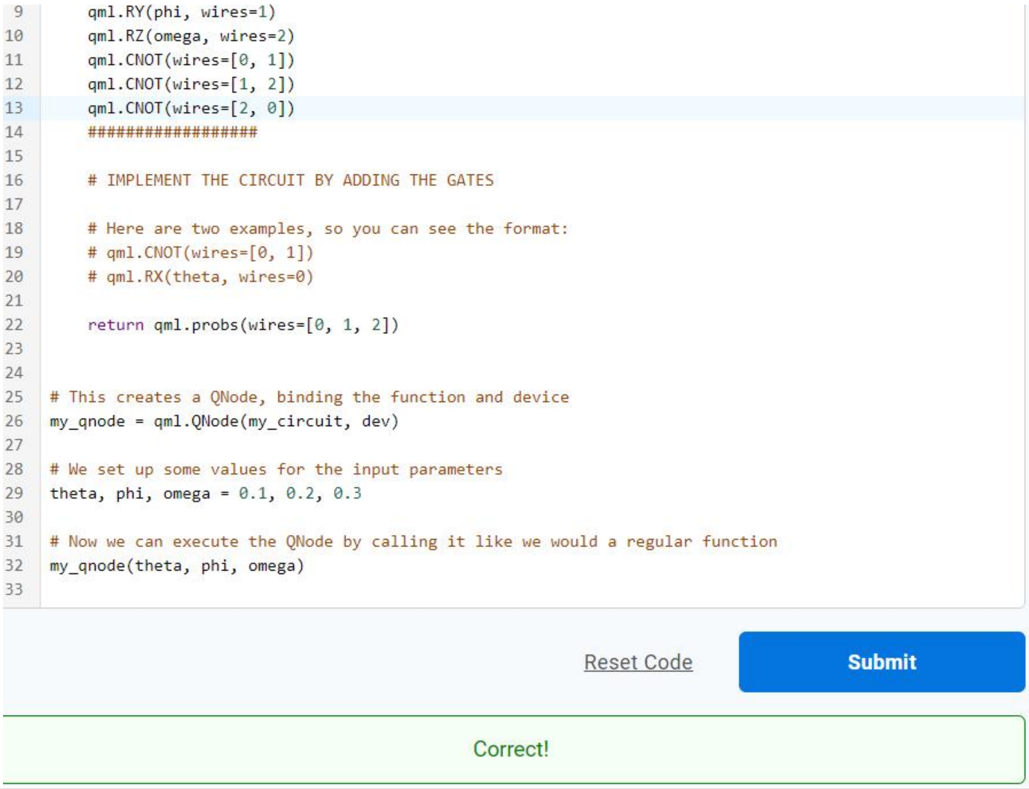
Complete the quantum function in the PennyLane code below to implement the following quantum circuit. Then, construct a QNode using `qml.QNode` and run the circuit on the provided device.



Solution:

```
# This creates a device with three wires on which PennyLane can run computations
dev = qml.device("default.qubit", wires=3)


def my_circuit(theta, phi, omega):

    ##################
    qml.RX(theta, wires=0)
    qml.RY(phi, wires=1)
    qml.RZ(omega, wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    ##################

    # IMPLEMENT THE CIRCUIT BY ADDING THE GATES

    # Here are two examples, so you can see the format:
    # qml.CNOT(wires=[0, 1])
    # qml.RX(theta, wires=0)

    return qml.probs(wires=[0, 1, 2])


# This creates a QNode, binding the function and device
my_qnode = qml.QNode(my_circuit, dev)
```

```
# We set up some values for the input parameters
theta, phi, omega = 0.1, 0.2, 0.3

# Now we can execute the QNode by calling it like we would a regular function
my_qnode(theta, phi, omega)
```



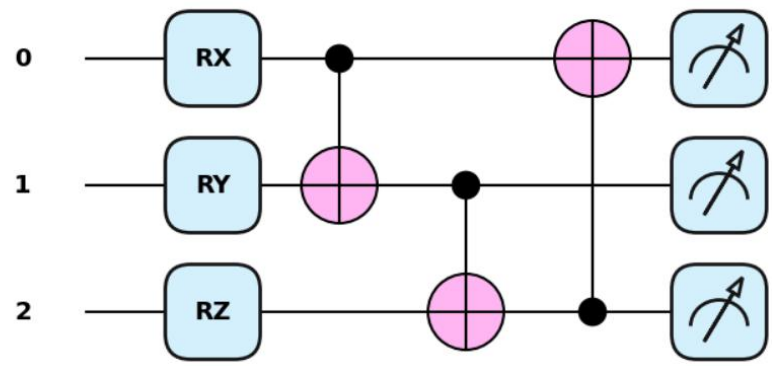
```
    qml.RY(phi, wires=1)
    qml.RZ(omega, wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    ##################

    # IMPLEMENT THE CIRCUIT BY ADDING THE GATES

    # Here are two examples, so you can see the format:
    # qml.CNOT(wires=[0, 1])
    # qml.RX(theta, wires=0)

    return qml.probs(wires=[0, 1, 2])


# This creates a QNode, binding the function and device
my_qnode = qml.QNode(my_circuit, dev)

# We set up some values for the input parameters
theta, phi, omega = 0.1, 0.2, 0.3

# Now we can execute the QNode by calling it like we would a regular function
my_qnode(theta, phi, omega)
```


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=3)

@qml.qnode(dev)
def my_circuit(theta, phi, omega):

    ##################
    qml.RX(theta, wires=0)
    qml.RY(phi, wires=1)
    qml.RZ(omega, wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    ##################

    # IMPLEMENT THE CIRCUIT BY ADDING THE GATES

    # Here are two examples, so you can see the format:
    # qml.CNOT(wires=[0, 1])
```

```
    # qml.RX(theta, wires=0)

    return qml.probs(wires=[0, 1, 2])


# This creates a QNode, binding the function and device
my_qnode = qml.QNode(my_circuit, dev)

# We set up some values for the input parameters
theta, phi, omega = 0.1, 0.2, 0.3

# Now we can execute the QNode by calling it like we would a regular function
my_qnode(theta, phi, omega)

qml.drawer.use_style("pennylane")
probs = qml.draw_mpl(my_qnode)(theta, phi,omega)
plt.show()
```

O/P:



Codercise I.2.3 — The QNode decorator

Open related theory

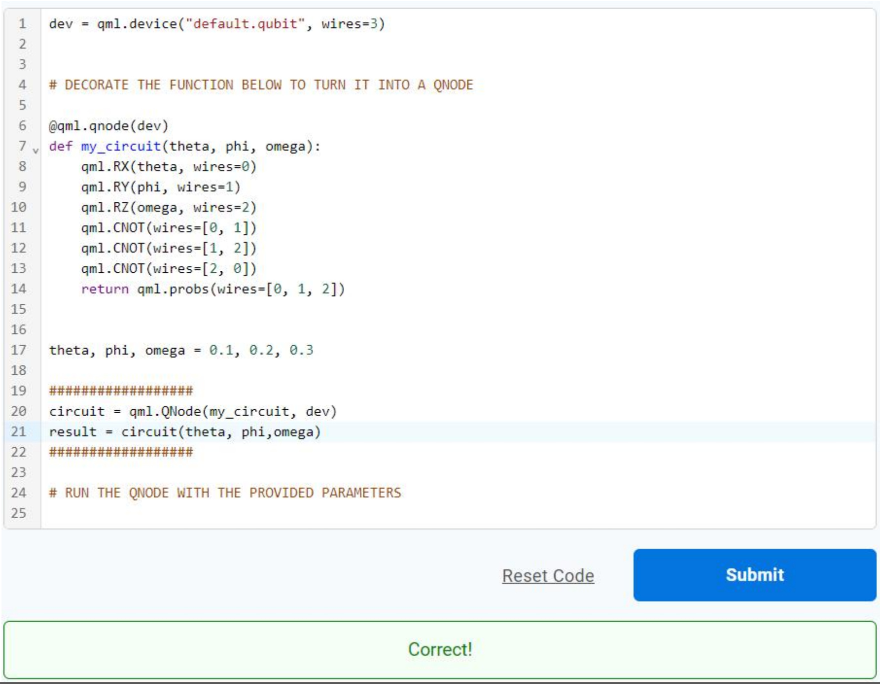
The second way to construct a QNode in PennyLane is using a **decorator**. Decorating a quantum function with `@qml.qnode(dev)` will automatically produce a QNode with the same name as your function that can be run on the device `dev`.

The quantum function below implements the circuit from the previous exercise. Apply a decorator to the quantum function to construct a QNode, then run it using the provided input parameters.

Solution:

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

```
# DECORATE THE FUNCTION BELOW TO TURN IT INTO A QNODE

@qml.qnode(dev)
def my_circuit(theta, phi, omega):
    qml.RX(theta, wires=0)
    qml.RY(phi, wires=1)
    qml.RZ(omega, wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    return qml.probs(wires=[0, 1, 2])


theta, phi, omega = 0.1, 0.2, 0.3

##################
circuit = qml.QNode(my_circuit, dev)
result = circuit(theta, phi,omega)
##################

# RUN THE QNODE WITH THE PROVIDED PARAMETERS
```



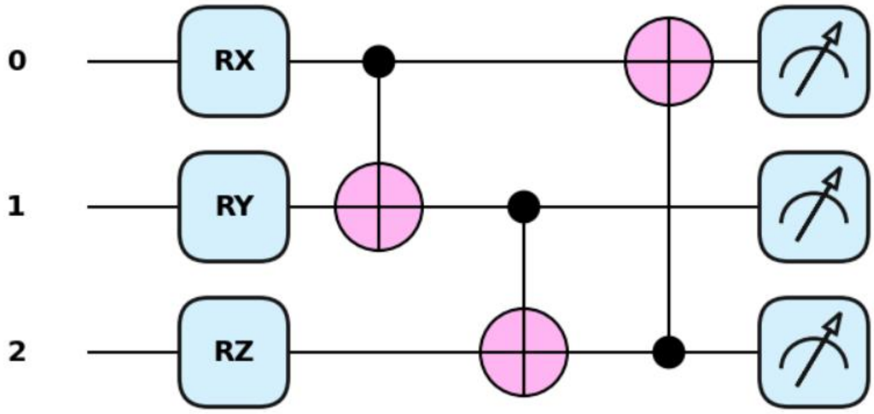
```
dev = qml.device("default.qubit", wires=3)


# DECORATE THE FUNCTION BELOW TO TURN IT INTO A QNODE

@qml.qnode(dev)
def my_circuit(theta, phi, omega):
    qml.RX(theta, wires=0)
    qml.RY(phi, wires=1)
    qml.RZ(omega, wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    return qml.probs(wires=[0, 1, 2])


theta, phi, omega = 0.1, 0.2, 0.3

#################
circuit = qml.QNode(my_circuit, dev)
result = circuit(theta, phi,omega)
#################

# RUN THE QNODE WITH THE PROVIDED PARAMETERS
```


Qiskit Program:

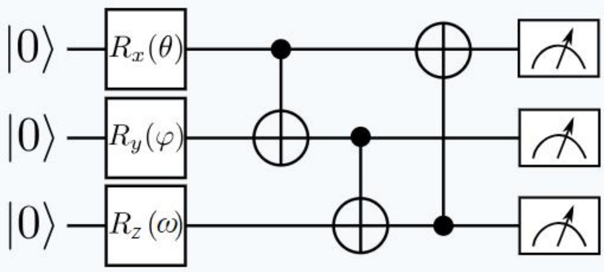
```
import numpy as np
import random
from qiskit.quantum_info import Statevector
import pennylane as qml
import matplotlib.pyplot as plt

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

@qml.qnode(dev)
def my_circuit(theta, phi, omega):

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

```
    qml.RX(theta, wires=0)
    qml.RY(phi, wires=1)
    qml.RZ(omega, wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    ##################

    # IMPLEMENT THE CIRCUIT BY ADDING THE GATES

    # Here are two examples, so you can see the format:
    # qml.CNOT(wires=[0, 1])
    # qml.RX(theta, wires=0)

    return qml.probs(wires=[0, 1, 2])


# We set up some values for the input parameters
theta, phi, omega = 0.1, 0.2, 0.3

circuit = qml.QNode(my_circuit, dev)
qml.drawer.use_style("pennylane")
result = qml.draw_mpl(circuit)(theta, phi,omega)
plt.show()
```

O/P



Remember our circuit from the previous section:



What is the depth of the circuit in the picture above?

Solution:

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


@qml.qnode(dev)
def my_circuit(theta, phi, omega):
    qml.RX(theta, wires=0)
    qml.RY(phi, wires=1)
    qml.RZ(omega, wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    return qml.probs(wires=[0, 1, 2])


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

# FILL IN THE CORRECT CIRCUIT DEPTH
depth = 4
```

Reset Code

Submit

Correct!

Qiskit Program:

```python
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=3)

@qml.qnode(dev)
def my_circuit(theta, phi, omega):

    ##################
    qml.RX(theta, wires=0)
    qml.RY(phi, wires=1)
    qml.RZ(omega, wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    ##################

    # IMPLEMENT THE CIRCUIT BY ADDING THE GATES

    # Here are two examples, so you can see the format:
```

```
    # qml.CNOT(wires=[0, 1])
    # qml.RX(theta, wires=0)

    return qml.probs(wires=[0, 1, 2])


# We set up some values for the input parameters
theta, phi, omega = 0.1, 0.2, 0.3

circuit = qml.QNode(my_circuit, dev)
specs_func = qml.specs(circuit)
specs_func(theta, phi,omega)
```

```
{'resources': Resources(num_wires=3, num_gates=6, gate_types=defaultdict(<class 'int'>, {'RX': 1, 'RY': 1, 'RZ': 1, 'CNOT': 3}), gate_sizes=defaultdict(<class 'int'>, {1: 3, 2: 3}), depth=4, shots=Shots(total_shots=None, shot_vector=())),
 'errors': {},
 'num_observables': 1,
 'num_diagonalizing_gates': 0,
 'num_trainable_params': 0,
 'num_device_wires': 3,
 'device_name': 'default.qubit',
 'expansion_strategy': 'gradient',
 'gradient_options': {},
 'interface': 'auto',
 'diff_method': 'best',
 'gradient_fn': 'backprop'}
```