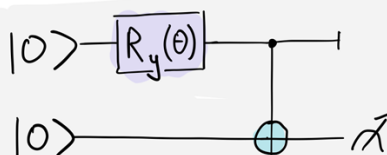## Simple Circuits: Entanglement [50 points]

Version: 1

---

**NOTE**: Coding templates are provided for all challenge problems at this link.
You are strongly encouraged to base your submission off the provided templates.

---

### Overview: Simple Circuit challenges

The **Simple Circuit** Challenges introduce quantum computation and PennyLane.
If you've never evaluated a quantum circuit before, this is a great place to start.
Even if you are familiar with quantum computation, this challenge will introduce
you to PennyLane circuits.

Central to PennyLane is the **QNode**, or Quantum Node. This object combines:

- A device
- A quantum function
- Any additional configuration information

We first need a **device**. The device evaluates a quantum function. It could
be either a simulator or actual quantum hardware. For the hackathon, we
recommend using `"default.qubit"`, a simple built-in simulator that does not
require external dependencies. To initialize a `"default.qubit"` device, we also
need to specify the number of wires.

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

Gates and measurements act on **wires**. Usually, a wire is just a qubit, a basic unit of quantum information storage. We use the more general term "wire" to encompass the quantum modes of photonic systems like in PennyLane's sister package Strawberry Fields. Any hashable object can label a wire, but people often denote individual wires by numbers (0, 1, 2) or strings ("alice", "auxiliary").

**Quantum functions** accept classical input, apply quantum operations, and return one or more quantum measurement statistics. We write quantum functions as plain, old Python functions, but with some constraints; the function always needs to return a **measurement**.

We can put these components together to see the required structure of a quantum function:

```python
def my_quantum_function(param):
    qml.PauliZ(wires=0) # a single-wire gate
    qml.RX(param, wires=0) # a single-wire parameterized gate
    qml.CNOT(wires=[0, 1]) # a two-wire gate

    # Finally we return a measurement of an operator on a wire
    return qml.expval(qml.PauliX(0))
```

While quantum functions look like Python functions, you can't call `my_quantum_function` on its own and have the described circuit executed. We instead have to call a QNode, an object containing both a quantum function and a device.

To quickly create a qnode, we apply a *decorator* that modifies the quantum function. The decorator takes the function as input and spits out something new that we can evaluate.

```python
@qml.qnode(dev)
def my_quantum_function(param):
    qml.PauliZ(wires=0) # a single-wire gate
    qml.RX(param, wires=0) # a single-wire parameterized gate
    qml.CNOT(wires=[0, 1]) # a two-wire gate

    # Finally we return a measurement of an operator on a wire
    return qml.expval(qml.PauliX(0))


result = my_quantum_function(0.1)
```

And that's it! Congratulations!

**More information**

To get more detail and examples, check out the documentation at these pages:

- Introduction

## Problem statement [50 points]

You are tasked with calculating a tensor-product observable for an entangled state. The provided template file `simple_circuits_50_template.py` contains a function `simple_circuits_50` that you need to complete.

As in the other "Simple Circuit" challenges, the `simple_circuits_50` function should:

- define a device
- create a quantum function and qnode
- execute the qnode

The circuit should:

1. Create the entangled Bell State:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

2. Rotate the first qubit around the y-axis by the provided angle using 'qml.RY'

3. Measure the tensor observable `qml.PauliZ(0) @ qml.PauliZ(1)`

We can draw the requested circuit using `qml.draw`:

```
0: --H--+C--RY(angle)--+| <Z @ Z>
1: -----+X------------+| <Z @ Z>
```

For those that prefer to see the mathematics, we can write the problem as:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

$$|\phi\rangle = R_{y,0}(\phi)|\Phi^+\rangle = \mathrm{e}^{-i(\sigma_y \otimes I)\phi/2}|\Phi^+\rangle$$

$$\mathrm{ans} = \langle\phi|\sigma_z^0 \otimes \sigma_z^1|\phi\rangle$$

**Input**

The `simple_circuits_50` function receives one float.

**Output**

The `simple_circuits_50` function must return a float.

**Acceptance Criteria**

In order for your submission to be judged as "correct":

- The outputs generated by your solution when run with a given `.in` file must match those in the corresponding `.ans` file to within the `Tolerance` specified below.

- Your solution must take no longer than the `Time limit` specified below to produce its outputs.

You can test your solution by passing the `#.in` input data to your program as stdin and comparing the output to the corresponding `#.ans` file:

```
python3 simple_circuits_50_template.py < 1.in
```

---

WARNING: Don't modify any of the code in the template outside of the `# QHACK #` markers, as this code is needed to test your solution. Do not add any print statements to your solution, as this will cause your submission to fail

---

---

Specs

---

Tolerance: **0.001 (0.1%)**
Timelimit: **30 s**

---

**Version History**

Version 1: Initial document.