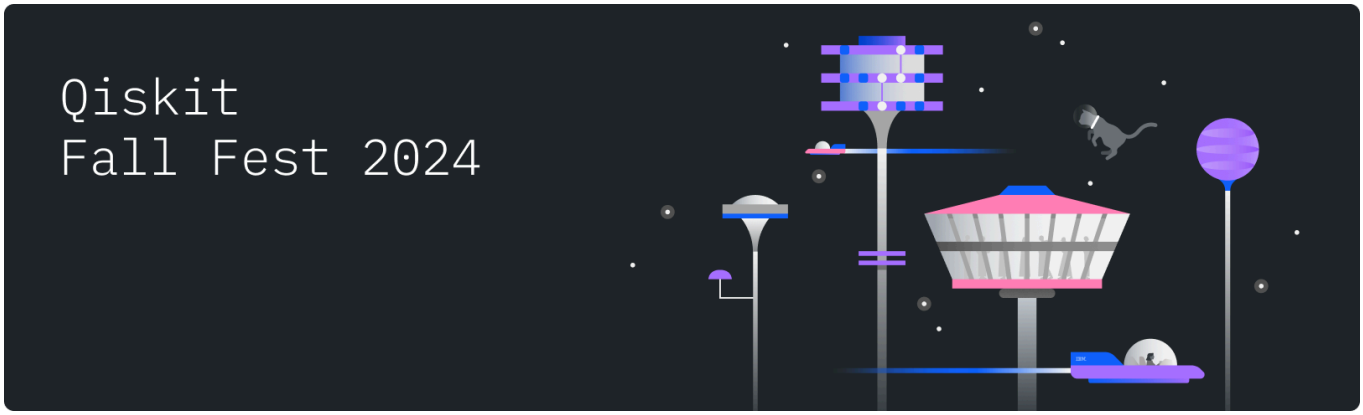


Graded Notebook 2: Qiskit Runtime Lab

- Difficulty: 3/5
- Estimated Time: 60 minutes

Hi there and welcome to the **Qiskit Fall Fest**!

Whether you're a total beginner or a PhD candidate in quantum physics, there is something here for you. Thanks for joining us. 😀

As part of the Qiskit Fall Fest, IBM Quantum has created a series of notebooks for you to work through, which include coding challenges and Qiskit tutorials.

The Qiskit Fall Fest is a massive event, featuring thousands of students worldwide who are all learning about quantum computing and Qiskit. Just by being here, you're helping to make history. Your participation is helping to shape what the future of the industry will look like. Congratulations and welcome!

Each of these notebooks builds upon the previous learning. This first one is meant for anyone to complete, even beginners, but later notebooks are more difficult. Most participants will need to do some outside research or use a bit of trial-and-error to finish the code challenges presented in the more advanced notebooks. Don't give up! We know you can do it.

In each notebook, you will find links to documentation, tutorials, and other helpful resources you might need to solve that particular problem. You can also find most of these resources on IBM's new home for quantum education: IBM Quantum Learning.

Table of Contents

Setup

Let's begin by getting all the necessary installs and imports out of the way.

If you are running things locally, you may already have these installed from a previous notebook. If you're running in the cloud, you may need to re-install for each notebook.

```
### Install Qiskit, if needed
```

```
%pip install qiskit[visualization]
%pip install qiskit_aer
%pip install qiskit_ibm_runtime
%pip install matplotlib
%pip install pylatexenc
%pip install qiskit-transpiler-service
%pip install git+https://github.com/qiskit-community/Quantum-Challenge-Grader.git
%pip install symengine==0.11.0
```

```
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.6.0->i
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.6.0->ipycyto
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.6.0->ipycytos
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.6.0->ipycyt
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython>
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.6.0->ipycytosc
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.6.0->ipycytosc
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.6.0->
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.6.0->ipycyt
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.10/dist-packages (from widgetsnbextension~=3.6.0->ipywidgets
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython>=4.0.0->ipywi
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ip
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.
Requirement already satisfied: jupyter-core>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextensi
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->
Requirement already satisfied: nbconvert>=5 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-packages (from jupyter-client->ipykernel>=4.5.1
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython>=4.0.0->ipywidge
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0-
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.6.1->notebook>=4.4.
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7->notebook>=4.4.
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextensio
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgets
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextens
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbex
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->wid
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->wi
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widget
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widg
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widget
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->w
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbexte
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat->notebook>=4.4.1->widge
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat->notebook>=4.4.1->widgetsnbe
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.1->jupyter-client->ipyker
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-packages (from argon2-cffi->notebook>=4.4.1->wi
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->notebook>=4.
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->notebo
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->notebook>=4
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/python3.10/dist-packages (from notebook-shim>=0.2.3->nbclassi
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings->argon2-cffi->notebo
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert>=5->notebook>
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert>=5->notebook>=4.4.1->w
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-c
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shim
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shi
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->no
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8-
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.
Requirement already satisfied: symengine==0.11.0 in /usr/local/lib/python3.10/dist-packages (0.11.0)
```

Save API Token, if needed

```
%set_env QXToken=18f7f21d5f05943b403bfcf461311b7347513ad32677f1657ffce2b612cd1a275c7ffa34d8c4d4b303e33b93893bf30e13c50abb7d75b58d221d089fabc

# Make sure there is no space between the equal sign
# and the beginning of your token
```

```
env: QXToken=18f7f21d5f05943b403bfcf461311b7347513ad32677f1657ffce2b612cd1a275c7ffa34d8c4d4b303e33b93893bf30e13c50abb7d75b58d221d089fabc
```

```
# imports
import numpy as np
```

```
from typing import List, Callable
from scipy.optimize import minimize
from scipy.optimize._optimize import OptimizeResult
import matplotlib.pyplot as plt

from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector, Operator, SparsePauliOp
from qiskit.primitives import StatevectorSampler, PrimitiveJob
from qiskit.circuit.library import TwoLocal
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit.visualization import plot_histogram
from qiskit_ibm_runtime.fake_provider import FakeSherbrooke
from qiskit_ibm_runtime import Session, EstimatorV2 as Estimator
from qiskit_aer import AerSimulator
```

This next cell is totally new. You need to import the grader information directly inside of each notebook.

```
# Setup the grader
from qc_grader.challenges.fall_fest24 import (
    grade_lab2_ex1,
    grade_lab2_ex2,
    grade_lab2_ex3,
    grade_lab2_ex4,
    grade_lab2_ex5,
    grade_lab2_ex6,
    grade_lab2_ex7,
)
```

Introduction

This notebook is a gentle introduction to primitives of Qiskit 1.x version. This notebook leverages some the latest innovations from IBM Quantum, such as Qiskit Runtime, and will help you get up to speed on the following topics and skills:

- How to set up quantum states using Qiskit
- Leveraging Qiskit Runtime Sampler and Estimator primitives, and Runtime
- Executing circuits in quantum simulators as well as on IBM Quantum computers by using Runtime

Part I: Qiskit states, the new and the old with Sampler

Subtitle: "Life is like a box of chocolates"

Now it time to get your hands dirty with Python and Qiskit code. Your assignment, should you choose to accept it, will be to develop a program that samples one piece from a box that contains two chocolate candies - vanilla ('01') and raspberry ('10') flavors. Your mission is build a quantum circuit that has a same probability of getting '01' and '10' by using a 'bell state'.

There exist 4 different Bell states. You can learn about each from the Basics of Quantum Information page

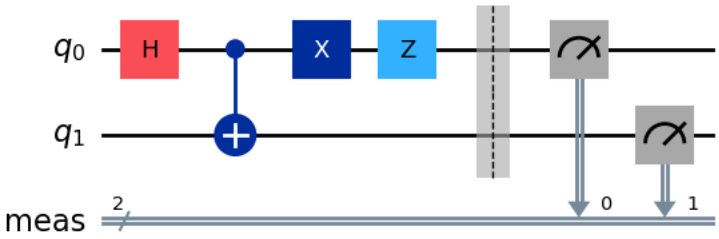
Exercise 1: Create and draw a singlet Bell state circuit

Bell circuits are specific circuits which generate Bell states, or EPR pairs, a form of entangled and normalized basis vectors. In other words, they are the circuits we use to generate entangled states, a key ingredient in quantum computations.

Your Task: please build a circuit that generates the $|\psi^-\rangle$ Bell state.

```
# Build a circuit to form a psi-minus Bell state
# Apply gates to the provided QuantumCircuit, qc

qc = QuantumCircuit(2)

### Write your code below here ###

# Apply Hadamard gate to the first qubit
qc.h(0)

# Apply CNOT gate with first qubit as control and second as target
qc.cx(0, 1)
```

```
# Apply X gate (NOT gate) to the first qubit
qc.x(0)

# Apply Z gate to the first qubit
qc.z(0)

### Don't change any code past this line ###
qc.measure_all()
qc.draw('mpl')
```



Hint: The $|\psi^-\rangle$ Bell state uses a single Z gate, and a single X gate, both of which occur after the CNOT.

```
# Submit your answer using following code

grade_lab2_ex1(qc) # Expected result type: QuantumCircuit
```

```
Grading your answer. Please wait...

Congratulations 🎉! Your answer is correct.
```

To observe the resulting entangled state that this circuit produces, we'll run our circuit a number of times and collect statistics on the final qubit measurements. That is the goal of the next exercise.

Exercise 2: Use Sampler.run

The Qiskit Sampler primitive (more info on Primitives here) returns the sampled result according to the specified output type. It allows us to efficiently sample quantum states by executing quantum circuits and providing probability distributions of the quantum states.

Your Task: use the Qiskit StatevectorSampler to obtain the counts resulting from our circuit.

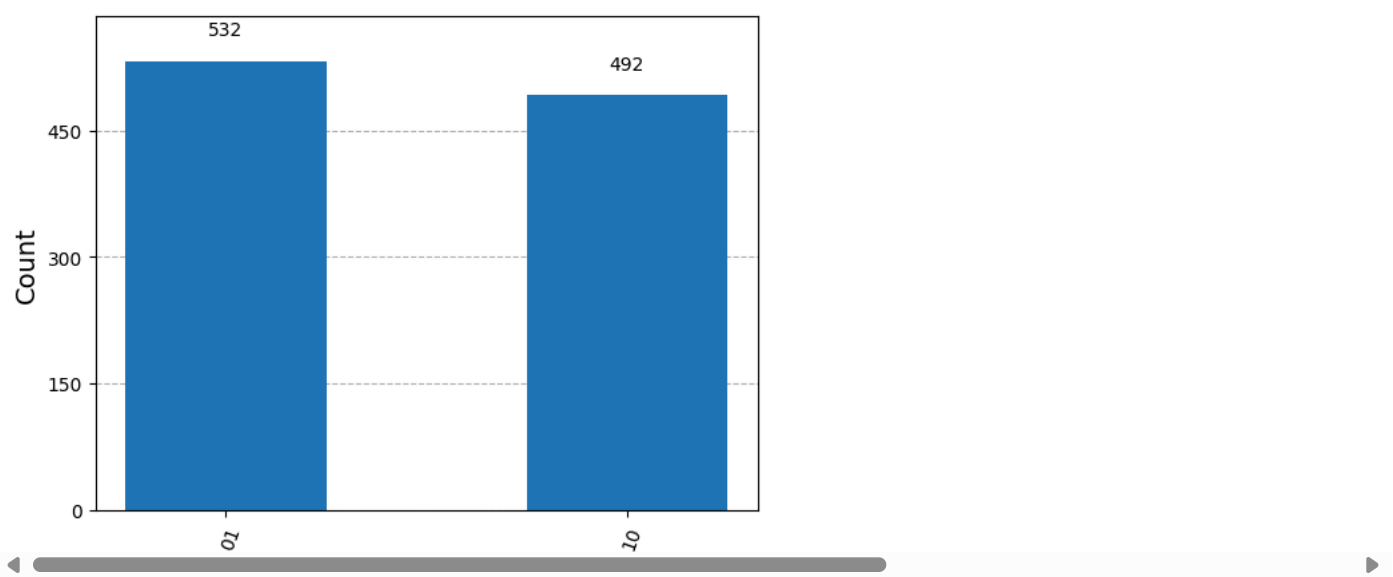
```
qc.measure_all()

### Write your code below here ###

from qiskit.primitives import StatevectorSampler
sampler = StatevectorSampler()

pub = (qc, [])
job_sampler = sampler.run([pub])

### Don't change any code past this line ###

result_sampler = job_sampler.result()
counts_sampler = result_sampler[0].data.meas.get_counts()

print(counts_sampler)
```

```
{'01': 532, '10': 492}
```

```
# Submit your answer using following code

grade_lab2_ex2(job_sampler) # Expected result type: PrimitiveJob
```

```
Grading your answer. Please wait...

Congratulations 🎉! Your answer is correct.
```

```
plot_histogram(counts_sampler)
```



The histogram shows an approximately even probability of finding our qubits in the `01` and the `10` states, suggesting that entanglement was performed as expected.

Exercise 3: Create and draw a W-state circuit

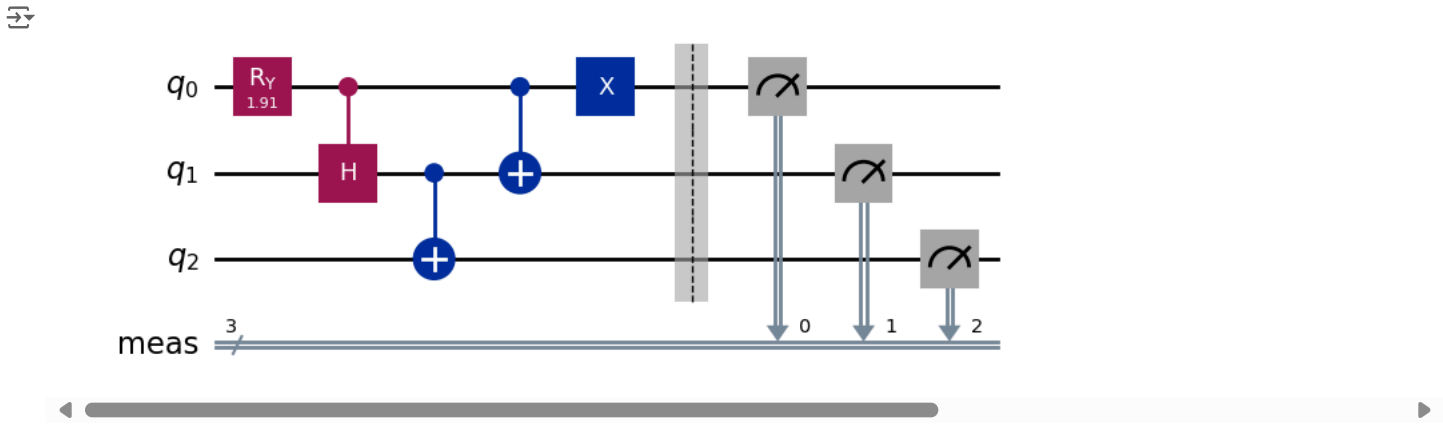
Next, we will develop a program samples one piece of candy from a box that contains three chocolate candies - Vanilla (`001`), Raspberry (`010`) and Strawberry (`001`). Your mission is build a circuit that has a equal probability of returning those chocolates. For this, we will use a **W-state circuit**.

Similarly to Bell states circuit producing Bell states, W-state circuits produce W states. Although Bell states entangle two qubits, W-states entangle three qubits. To build our W-state, we will follow 6 simple steps:

1. Initialize our 3 qubit circuit
2. Perform an Ry rotation on our qubit. The specifics of this operation are provided.
3. Perform a controlled hadamard gate on qubit 1, with control qubit 0
4. Add a CNOT gate with control qubit 1 and target qubit 2
5. Add a CNOT gate with control qubit 0 and target qubit 1
6. Add a X gate on qubit 0

Your Task: Follow the steps to build the W-state circuit

```
# Step 1
qc = QuantumCircuit(3)

# Step 2 (provided)
qc.ry(1.91063324, 0)

# Add steps 3-6 below

# Apply Hadamard gate to the first qubit
qc.ch(0, 1)

# Apply CNOT gate with first qubit as control and second as target
qc.cx(1, 2)
# Apply CNOT gate with zero qubit as control and first as target
qc.cx(0, 1)

# Add a X gate on qubit 0
qc.x(0)



### Don't change any code past this line ###
qc.measure_all()
qc.draw('mpl')
```




```
# Submit your answer using following code

grade_lab2_ex3(qc) # Expected result type: # Expected result type: QuantumCircuit
```

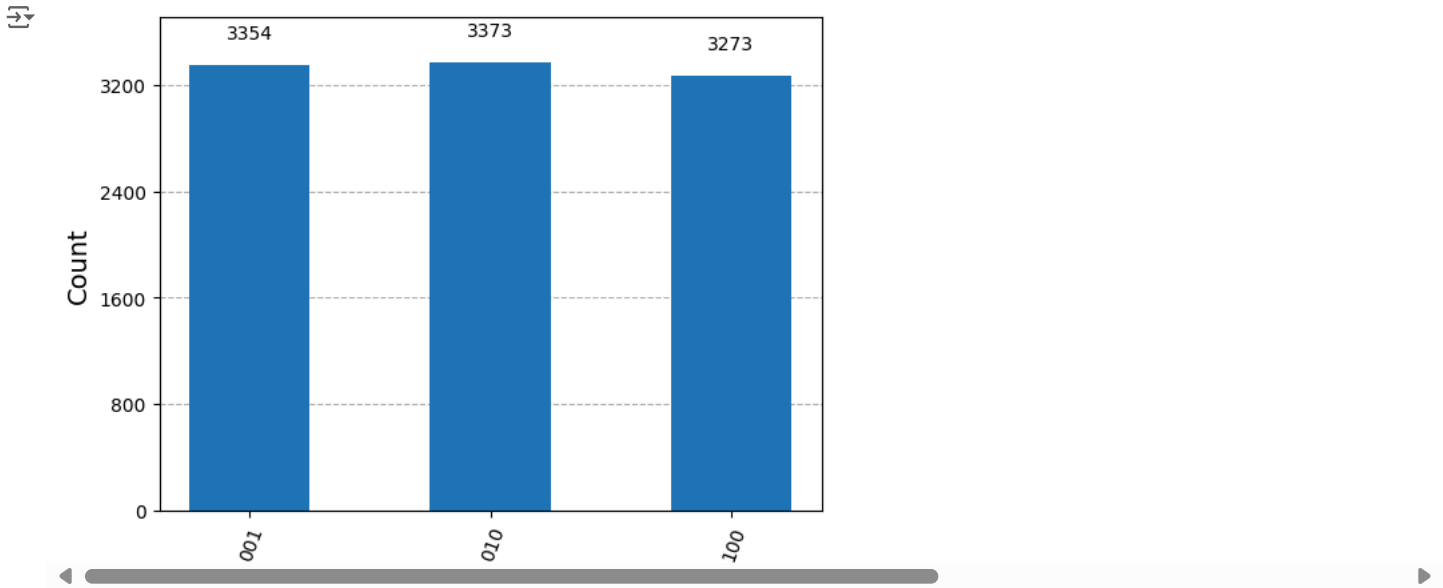
```
Grading your answer. Please wait...

Congratulations 🎉! Your answer is correct.
```

Once again, let's visualize our results:

```
sampler = StatevectorSampler()
pub = (qc)
job_sampler = sampler.run([pub], shots=10000)

result_sampler = job_sampler.result()
counts_sampler = result_sampler[0].data.meas.get_counts()

print(counts_sampler)
```

```
{'010': 3373, '001': 3354, '100': 3273}
```

```
plot_histogram(counts_sampler)
```



We observe our total counts distributed in three similarly sized bins in three states, which are the three desired states for the successful creation of a W-state. Good work!

Now that we have basic circuits down, we'll start introducing and developing more complex codes with Qiskit 1.0.

Part II: VQE with Qiskit 1.0

Subtitle: "You never know what you're gonna get"

When selecting a piece of chocolate candy from a box, the piece often meets our expectations. Sometimes after biting into the candy however, we find that the filling is not our favorite. This exercise leverages Qiskit Runtime and a Variational Quantum Eigensolver (VQE) to assemble a box of chocolates that hopefully meets our expectations. It uses the Qiskit Runtime Estimator to calculate the expectation values for combinations of candies in a box, and uses a Qiskit Runtime session to facilitate running a VQE algorithm to find the highest expectation value.

The core of this challenge will leverage Qiskit Runtime and a Variational Quantum Eigensolver (VQE). We will be using the Qiskit Runtime Estimator to calculate expectation values for combinations of qubits, and Qiskit Runtime Sessions to facilitate running a VQE algorithm. The challenge draws upon code from an example that experimental physicist & IBM Quantum researcher Nick Bronn created for the Coding with Qiskit Runtime video series, specifically in Episode 05 Primitives & Sessions, and implements it using the newest Qiskit 1.0 version.

Let's start by creating a Pauli operator

There are three pieces of chocolate candy in a box. Each piece can either have a vanilla center or a raspberry center. The user likes vanilla centers but dislikes raspberry centers, so we'll say that picking a vanilla it is worth 1 point, but picking a raspberry it is worth -1 point. To model this as an operator, we'll create three Pauli operators and sum them together. Each piece of candy in the box is represented by a qubit, and each of these Pauli's contain expectation values for their corresponding position in the box. Summing them together results in the diagonal of the operator containing eigenvalues that represent expectation values for all eight possible combinations of candy in the box.

A Pauli operator is a matrix representing a quantum mechanical observable corresponding to a measurement of spin along a particular axis (x, y, z). Let's build one by assuming you like all of the flavors - vanilla, raspberry and chocolate!

```
pauli_op = SparsePauliOp(['ZII', 'IZI', 'IIZ'])
print(pauli_op.to_matrix())
```

```
[[ 3.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  1.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  1.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j -1.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j  0.+0.j  1.+0.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j -1.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j -1.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j  0.+0.j -3.+0.j]]
```

Setup and run a VQE algorithm

Now, we'll start setting up our VQE algorithm. Variational quantum eigensolvers (VQEs), are hybrid algorithms that utilize quantum and classical techniques to find the ground state of a given physical system. They are often used in quantum chemistry and optimization problems, and are promising candidates for hybrid-algorithms in noisy near-term devices.

VQEs are characterized by the use of a classical optimization algorithm to iteratively improve upon a parameterized trial solution, called an "ansatz". The aim is to solve for the ground state of a given Hamiltonian represented as a linear combination of Pauli terms.

Executing a VQE algorithm requires these three steps:

1. Setting up the Hamiltonian and ansatz (problem specification)
2. Implementing the Qiskit Runtime estimator
3. Adding the Classical optimizer and running our program

We we will follow these steps.

Exercise 4: Create a parameterized circuit to serve as the ansatz

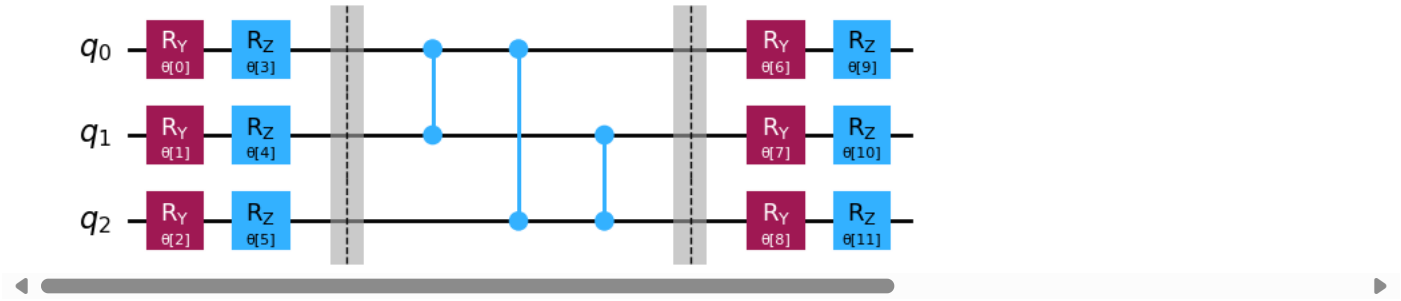
Our first task will be to set up our ansatz, or a trial solution, for our problem which we will compare against.

For this we can use Qiskit's `TwoLocal` circuit, a pre-built circuit that can be used to prepare trial wave functions for variational quantum algorithms or classification circuits for machine learning. TwoLocal circuits are parameterized circuits consisting of alternating rotation layers and entanglement layers. You can find more information about them in Qiskit's documentation.

Your Task: Set up a 3-qubit TwoLocal circuit using Ry and Rz rotations. Entanglement should be set to full, and entanglement blocks should use the Cz gate. Make sure you set `reps=1` and `insert_barriers=True`.

```
num_qubits = 3
rotation_blocks = ['ry', 'rz']
entanglement_blocks = 'cz'
entanglement = 'full'

ansatz = TwoLocal(num_qubits=num_qubits, rotation_blocks=rotation_blocks, entanglement_blocks=entanglement_blocks, entanglement=entanglement
```

```
### Don't change any code past this line ###
ansatz.decompose().draw('mpl')
```




```
# Submit your answer using following code

grade_lab2_ex4(num_qubits, rotation_blocks, entanglement_blocks, entanglement) # Expected result type: int, List[str], str, str
```

```
Grading your answer. Please wait...

Congratulations 🎉! Your answer is correct.
```

From the previous figure we see that our ansatz circuit is defined by a vector of parameters θ, with the total number given by:

```
num_params = ansatz.num_parameters
num_params
```

```
12
```

Exercise 5: Transpile to ISA circuits

In this example we will use the FakeSherbrooke, a fake (simulated) 127-qubit backend, useful for testing the transpiler and other backend-facing functionalities.

Preset pass managers are the default pass managers used by the `transpile()` function. `transpile()` provides a convenient and simple method to construct a standalone `PassManager` object that mirrors what the transpile function does when optimizing and transforming a quantum circuit for execution on a specific backend.

Your Task: Define the pass manager. Reference the Qiskit documentation for more info.

```
backend_answer = FakeSherbrooke()
optimization_level_answer = 0

pm = generate_preset_pass_manager(backend=backend_answer, optimization_level=optimization_level_answer)

# Apply the pass manager to the circuit
isa_circuit = pm.run(ansatz)

# isa_circuit.draw('mpl')
```

Tip: Make sure you are using the right backend!

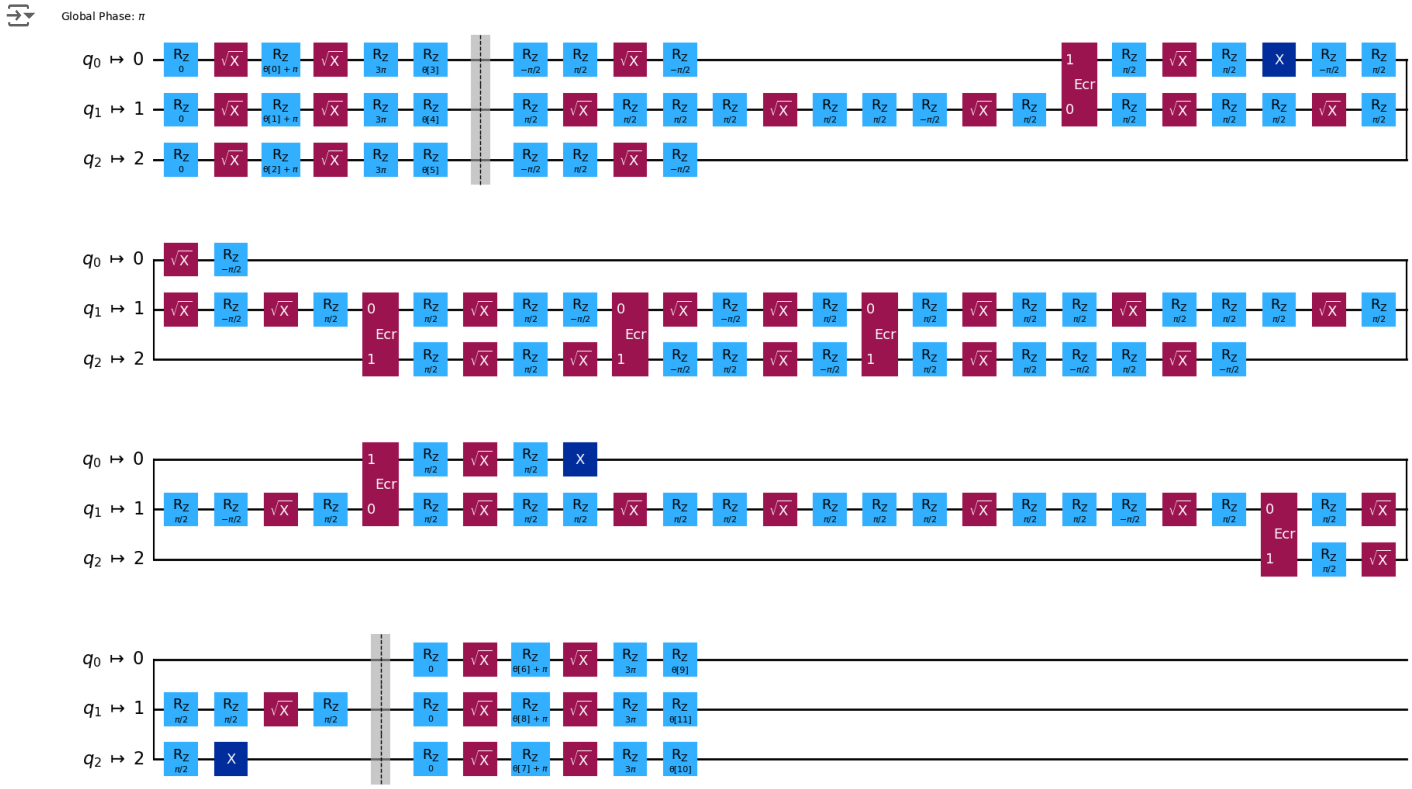
The grader was designed with **FakeSherbrooke** in mind, and therefore is expecting a 127 qubit map. If you use another backend you might encounter issues.

```
# Submit your answer using following code

grade_lab2_ex5(isa_circuit) # Expected result type: QuantumCircuit
```

```
Grading your answer. Please wait...

Congratulations 🎉! Your answer is correct.
```

With the newest version of Qiskit Runtime, all circuits submitted to a backend must conform to the constraints of the backend's Target. Such circuits are considered to be written in terms of that backend's Instruction Set Architecture (ISA) — i.e., the set of instructions the device can understand and execute. These Target constraints are defined by factors like the device's native basis gates, its qubit connectivity, and when relevant, its pulse and other instruction timing specifications. To visualize our ISA circuits we can run:

```
isa_circuit.draw('mpl', idle_wires=False,)
```



As you can see, after transpilation, the circuit only contain the native basis gates of the backend. For more details on ISA circuits check out these resources from the IBM team:

- What are ISA circuits?

- Understanding the new ISA circuits requirement

You can run the next cell to define our Hamiltonian, then move on to Exercise 6.

```
# Define our Hamiltonian
hamiltonian_isa = pauli_op.apply_layout(layout=isa_circuit.layout)
```

Exercise 6: Defining the cost function

Like many classical optimization problems, the solution to a VQE problem can be formulated as minimization of a scalar cost function. The cost function for our VQE is simple: the energy!

Your Task: Define a cost function by using Qiskit Runtime Estimator to find the energy for a given parameterized state and our Hamiltonian.

```
def cost_func(params, ansatz, hamiltonian, estimator, callback_dict):
    """Return estimate of energy from estimator

    Parameters:
        params (ndarray): Array of ansatz parameters
        ansatz (QuantumCircuit): Parameterized ansatz circuit
        hamiltonian (SparsePauliOp): Operator representation of Hamiltonian
        estimator (EstimatorV2): Estimator primitive instance

    Returns:
        float: Energy estimate
    """
    # creates a "primitive unified bloc" (pub) which is a tuple containing the ansatz circuit, the Hamiltonian operator, and the current par
    pub = (ansatz, [hamiltonian], [params])

    # runs the Estimator with our pub and retrieves the result
    result = estimator.run([pub]).result()

    # result.data is a list of PubResult objects (one for each pub submitted),
    # access the first PubResult with .evs gives us the expectation values
    energy = result[0].data.evs[0]

    callback_dict["iters"] += 1
    callback_dict["prev_vector"] = params
    callback_dict["cost_history"].append(energy)


### Don't change any code past this line ###
    print(energy)
    return energy, result


# Submit your answer using following code

grade_lab2_ex6(cost_func) # Expected result type: Callable
```

```
[0.64160156]
Grading your answer. Please wait...

Congratulations 🎉! Your answer is correct.
```

Callback functions are a standard way for users to obtain additional information about the status of an iterative algorithm (such as VQE). However, it is possible to do much more than this. Here, we use a mutable object (dictionary), to store resulting vector at each iteration of our algorithm, in case we need to restart the routine due to failure or return the another iteration number.

```
callback_dict = {
    "prev_vector": None,
    "iters": 0,
    "cost_history": [],
}
```

Using the Classical Optimizer

We can now use a classical optimizer of our choice to minimize the cost function. In real quantum hardware, the choice of optimizer is important, as not all optimizers handle noisy cost function landscapes equally well. Here, we can use SciPy routines.

To begin the routine, we specify a random initial set of parameters:

```
x0 = 2 * np.pi * np.random.random(num_params)
x0
```

```
array([3.63098301, 2.35767284, 2.38801562, 6.19608389, 5.14126734,
       6.24812409, 4.96054148, 1.37425685, 1.95932931, 3.90597346,
       3.24819083, 1.85173447])
```

Exercise 7: QiskitRuntimeService V2 Primitives, local testing mode, and Sessions

Next, we will use the new QiskitRuntimeService V2 primitives: EstimatorV2 and SamplerV2.

The new Estimator interface lets you specify a single circuit and multiple observables and parameter value sets for that circuit, so that sweeps over parameter value sets and observables can be efficiently specified. Previously, you had to specify the same circuit multiple times to match the size of the data to be combined. Also, while you can still use optimization_level and resilience_level as the simple knobs, V2 primitives give you the flexibility to turn on or off individual error mitigation / suppression methods to customize them for your needs.

SamplerV2 is simplified to focus on its core task of sampling the quantum register from the execution of quantum circuits. It returns the samples, whose type is defined by the program, without weights. The output data is also separated by the output register names defined by the program. This change enables future support for circuits with classical control flow.

We will also use Qiskit's 1.0 local testing mode. Local testing mode (available with qiskit-ibm-runtime 0.22.0 or later) can be used to help develop and test programs before fine-tuning them and sending them to real quantum hardware.

Your Task: Use local testing mode to verify your program, then change the backend name to run it on an IBM Quantum system.

```
### Select a Backend
## Use FakeSherbrooke to simulate with noise that matches closer to the real experiment. This will run slower.
## Use AerSimulator to simulate without noise to quickly iterate. This will run faster.

# backend = FakeSherbrooke()
backend = AerSimulator()

# ### Don't change any code past this line ###

# Here we have updated the cost function to return only the energy to be compatible with recent scipy versions (>=1.10)
def cost_func_2(*args, **kwargs):
    energy, result = cost_func(*args, **kwargs)
    return energy

estimator = Estimator(backend)

res = minimize(
    cost_func_2,
    x0,
    args=(isa_circuit, hamiltonian_isa, estimator, callback_dict),
    method="cobyla",
    options={'maxiter': 100})
```

```
-2.99462890625
-2.99658203125
-2.99609375
-2.99755859375
-2.99462890625
-2.9990234375
-2.998046875
-2.99853515625
-2.998046875
-2.990234375
-2.9970703125
-2.99853515625
-2.998046875
-2.994140625
-2.998046875
-2.9951171875
-2.99609375
-2.99755859375
-2.9970703125
-2.9970703125
-2.99658203125
-2.99951171875
-2.9970703125
-2.99609375
-2.99755859375
-2.99755859375
-2.99560546875
-2.9970703125
-2.998046875
-2.9970703125
-2.9970703125
-2.9951171875
-2.99658203125
```

```
# Submit your answer using following code

grade_lab2_ex7(res) # Expected result type: OptimizeResult
```

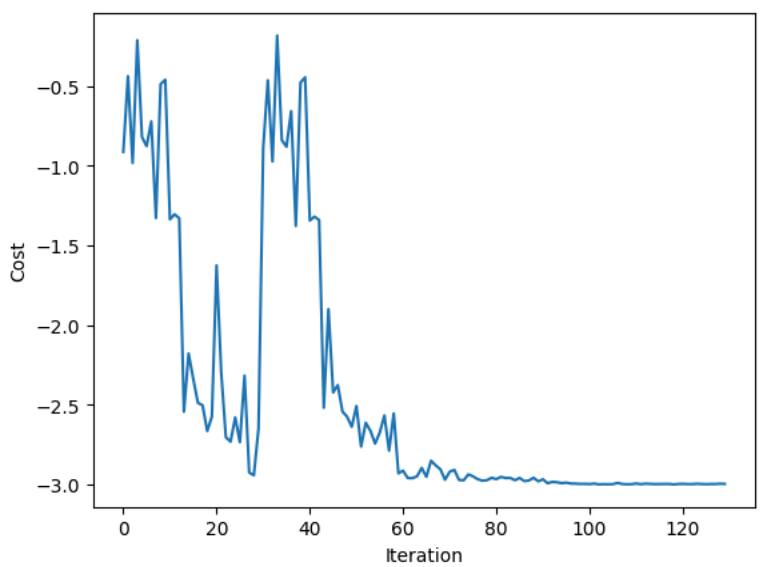
```
Grading your answer. Please wait...

Congratulations 🎉! Your answer is correct.
```

Tip: Increase `maxiter` if you do not see convergence

If the cost is not converging, increase the `maxiter` (100 is an appropriate number) and run Ex 7 again.

Let's look at our results:

```
fig, ax = plt.subplots()
plt.plot(range(callback_dict["iters"]), callback_dict["cost_history"])
plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.draw()
```



🎉 As expected our VQE algorithm minimized our energy across iterations, until reaching the ground state. We have now successfully implemented a VQE algorithm using brand-new Qiskit 1.0 functionalities!

Thank you for completing this notebook, and good luck with the remaining ones!

Additional information

Created by: James Weaver, Maria Gragera Garces

Advised by: Junye Huang

Version: 1.3.0