IBM Quantum Challenge Fall 2022

Challenge 1 - Introduction to Primitives on Qiskit Runtime

Table of Contents

Table of Contents

Lab 1 – Introduction to Primitives on Qiskit Runtime

Lab 2 – Quantum Kernel Learning with Qiskit Runtime

Lab 3 – Optimization with Qiskit Primitives

Lab 4 – Quantum Chemistry with Qiskit Primitives

Table of Contents

Lab 1 – Introduction to Primitives on Qiskit Runtime

Lab 2 – Quantum Kernel Learning with Qiskit Runtime

Lab 3 – Optimization with Qiskit Primitives

Lab 4 – Quantum Chemistry with Qiskit Primitives

Story

Introduction to Primitives

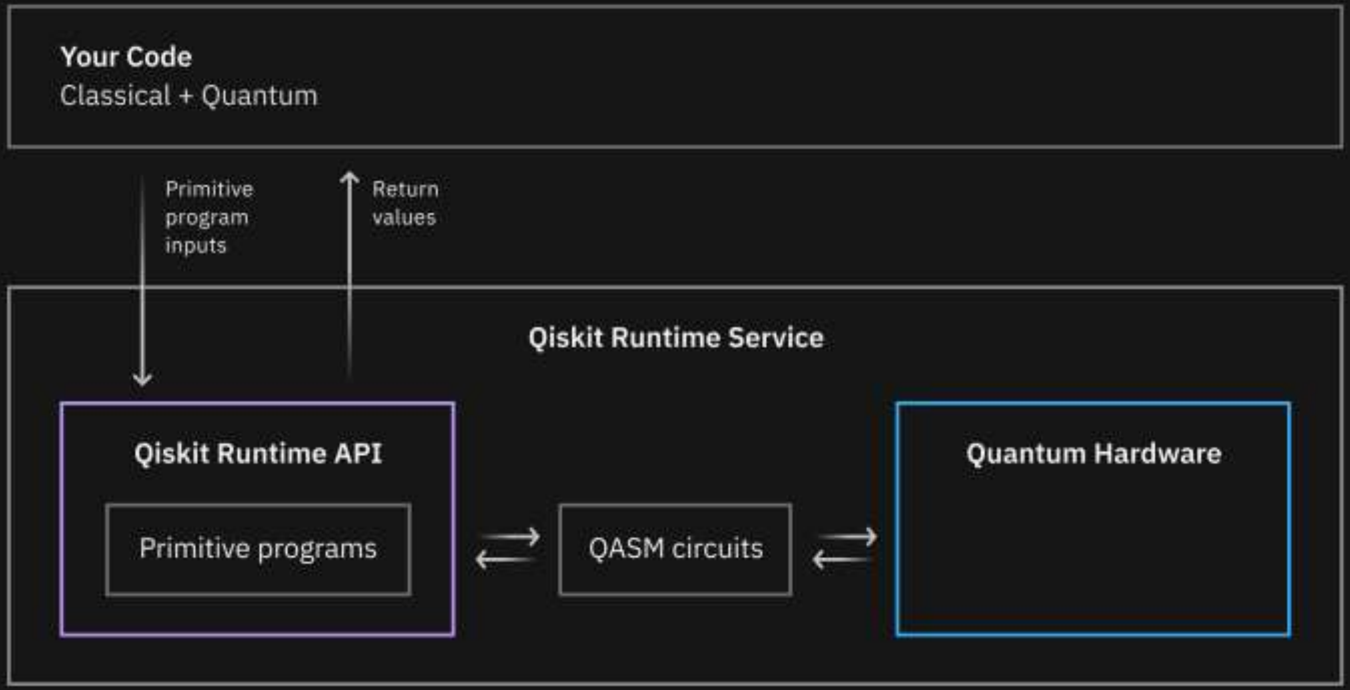
Basic Element

that serve as a building block for more complex elements

Primitives with Quantum?

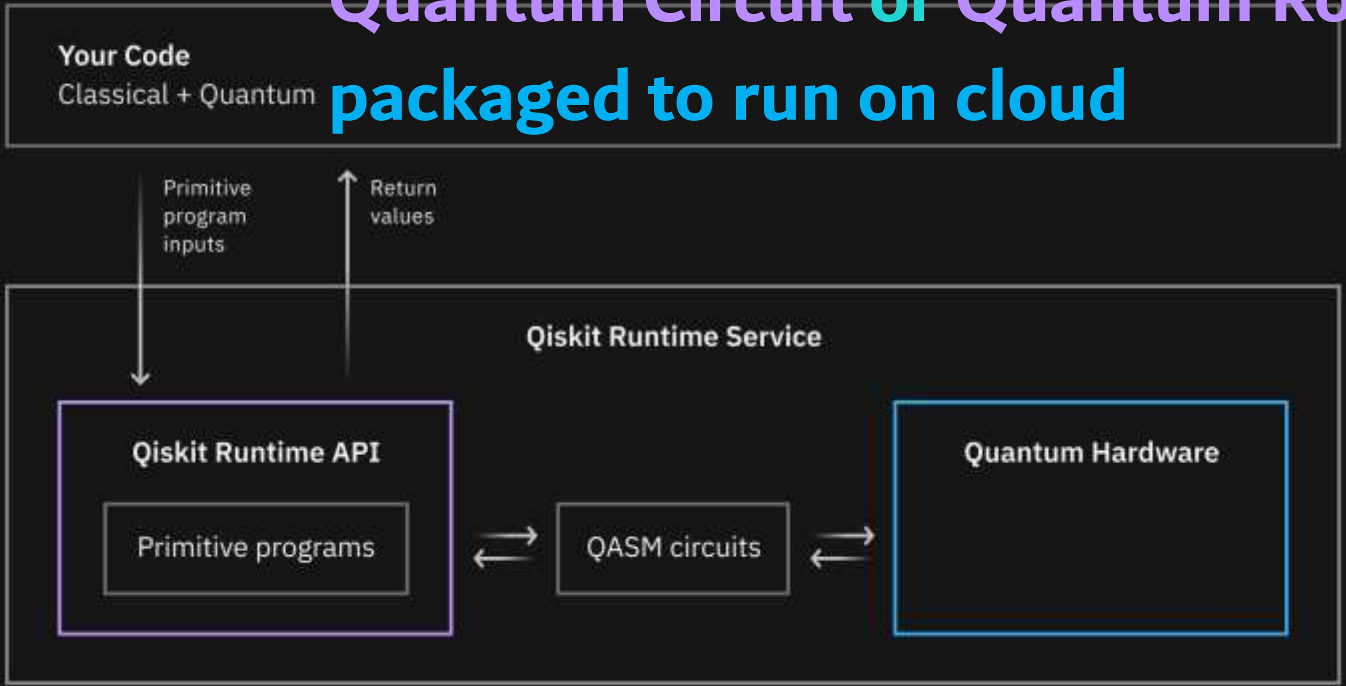
- Higher level core primitive terms with categorization
- Enable to build a program more accessibility
- Do not need low machine code level

Qiskit Runtime



Qiskit Runtime

Quantum Circuit or Quantum Routines are packaged to run on cloud



Your Code

Classical + Quantum

Primitive program inputs

Return values

Qiskit Runtime Service

Qiskit Runtime API

Primitive programs

QASM circuits

Quantum Hardware

Qiskit Runtime

Save Latency

Reduce overhead for iterative loops

Qiskit Runtime

Qiskit

Efficiency

Save Latency

Reduce overhead for iterative loops

Qiskit Runtime

Qiskit

Efficiency

Consistency

Save Latency

Reduce overhead for iterative loops

Qiskit Runtime

Qiskit

Efficiency **Consistency** **Customizability**

Save Latency

Reduce overhead for iterative loops

Qiskit Runtime

Efficiency **Consistency** **Customizability**

Save Latency

Reduce overhead for iterative loops

Error Mitigation and Suppression

Qiskit Runtime keyword

- **Session**
  - Open and Close session to use the primitives on cloud
  - Define a job as collection of iterative calls to the quantum computer

- **Options**
  - Configure the current session and parameter
  - Control execution environment

Qiskit Runtime Primitive

Sampler

Estimator

Qiskit Runtime Sampler

- Generates an error-mitigated readout of quasiprobabilities
  - **Input**: User Circuit

- **Better** evaluate shot result
  - Error mitigation
  - More efficient evaluation of the possibility of multiple relevant data points
    - **Where**: the context of destructive interference

Bernstein-Vazirani algorithm

$$f(x) = s \cdot x (mod 2)$$

s, hidden string of bits

Bernstein-Vazirani algorithm

$$f(x) = s \cdot x (mod 2)$$

s, hidden string of bits

Classical Computer:

Call the function n times

n, length of hidden string

Bernstein-Vazirani algorithm

$$f(x) = s \cdot x(mod2)$$

s, hidden string of bits

Classical Computer:

Call the function n times

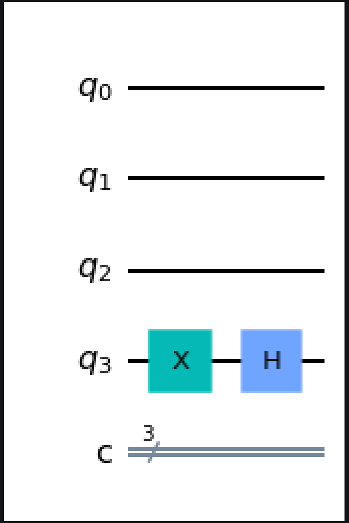
n, length of hidden string

Quantum Computer:

Call the function **ONE** time to solve with 100% confidence

Quantum Bernstein-Vazirani algorithm

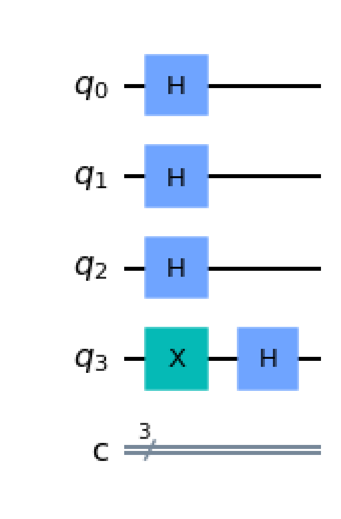
1. Initialize the input qubits to the state $|0\rangle^{\otimes n}$, and output qubit to $|-\rangle$



n=3

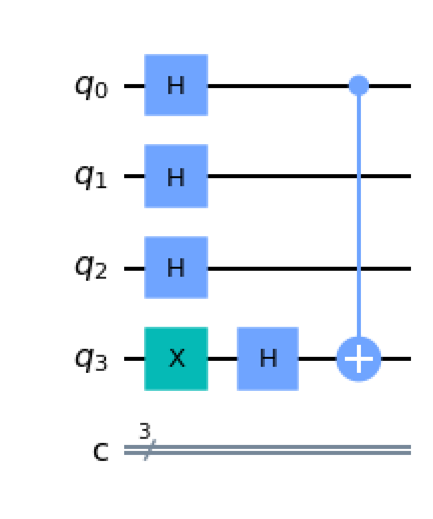
Quantum Bernstein-Vazirani algorithm

1. Initialize the input qubits to the state $|0\rangle^{\otimes n}$, and output qubit to $|-\rangle$
2. Apply Hadamard gates to the input register



Quantum Bernstein-Vazirani algorithm

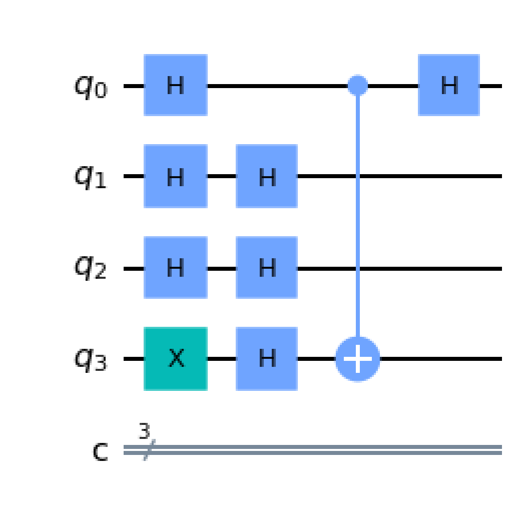
1. Initialize the input qubits to the state $|0\rangle^{\otimes n}$, and output qubit to $|-\rangle$
2. Apply Hadamard gates to the input register
3. Query the oracle.



Hidden circuit = “001”

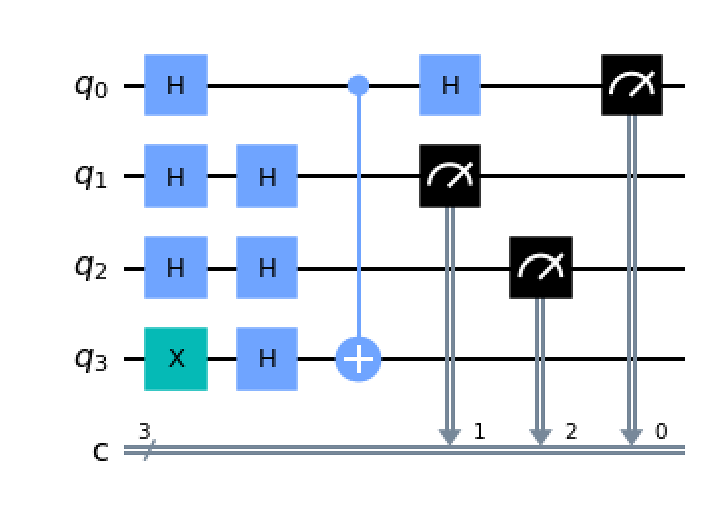
Quantum Bernstein-Vazirani algorithm

1. Initialize the input qubits to the state $|0\rangle^{\otimes n}$, and output qubit to $|-\rangle$
2. Apply Hadamard gates to the input register
3. Query the oracle.
4. Apply Hadamard gates to the input register.



Quantum Bernstein-Vazirani algorithm

1. Initialize the input qubits to the state $|0\rangle^{\otimes n}$, and output qubit to $|-\rangle$
2. Apply Hadamard gates to the input register
3. Query the oracle.
4. Apply Hadamard gates to the input register.
5. Measure input qubits

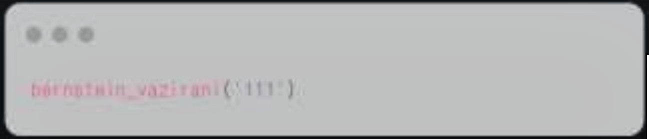


Exercise 1

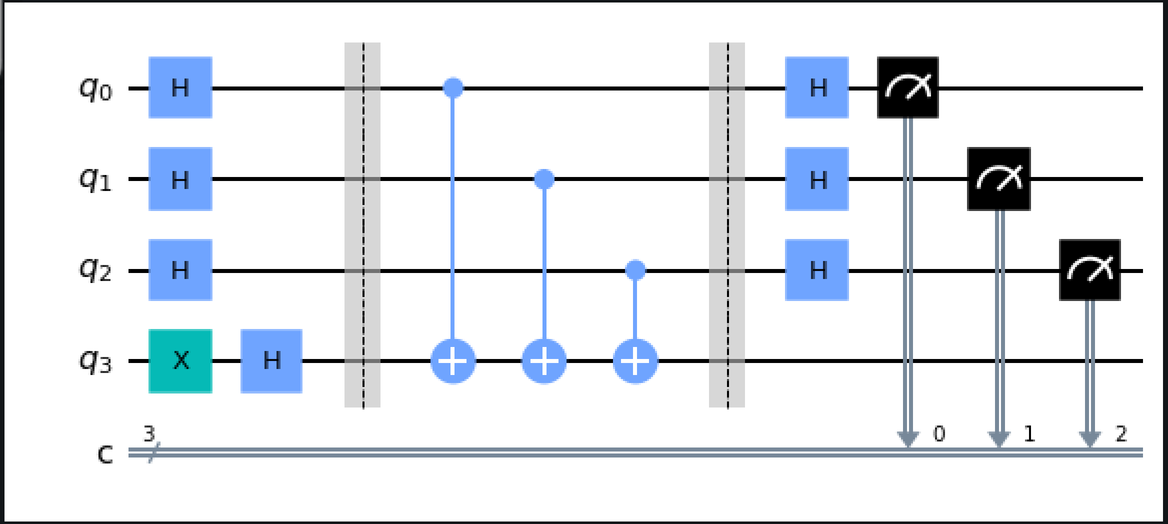
Build the `bernstein_vazirani` function

Exercise 1

Build the `bernstein_vazirani` function



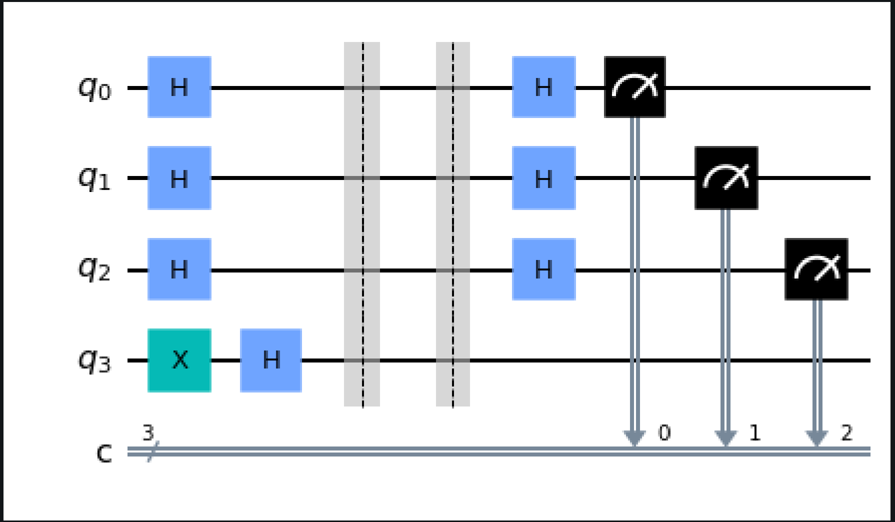
```
bernstein_vazirani('111')
```



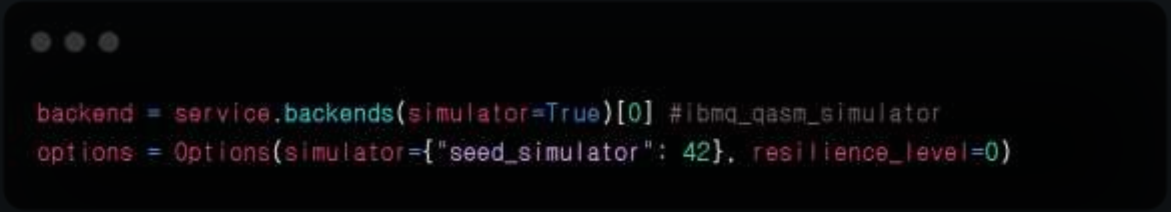
Exercise 1

Build the `bernstein_vazirani` function

```
bernstein_vazirani('000')
```

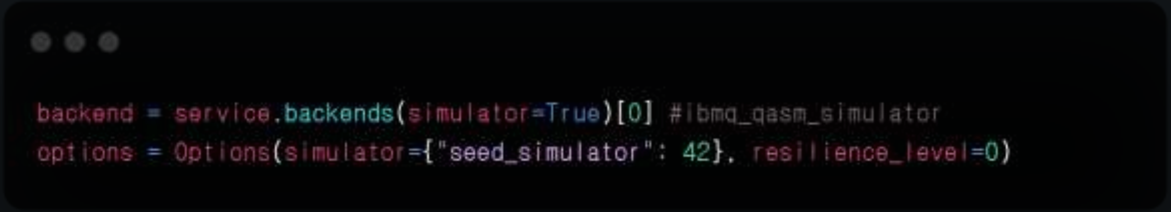


Use Runtime to run the circuit

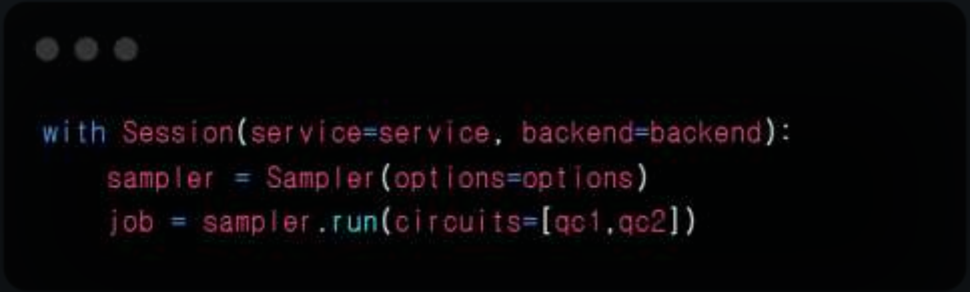


```
backend = service.backends(simulator=True)[0] #ibmq_qasm_simulator
options = Options(simulator={"seed_simulator": 42}, resilience_level=0)
```

Use Runtime to run the circuit



```
backend = service.backends(simulator=True)[0] #ibmq_qasm_simulator
options = Options(simulator={"seed_simulator": 42}, resilience_level=0)
```



```
with Session(service=service, backend=backend):
    sampler = Sampler(options=options)
    job = sampler.run(circuits=[qc1,qc2])
```

result

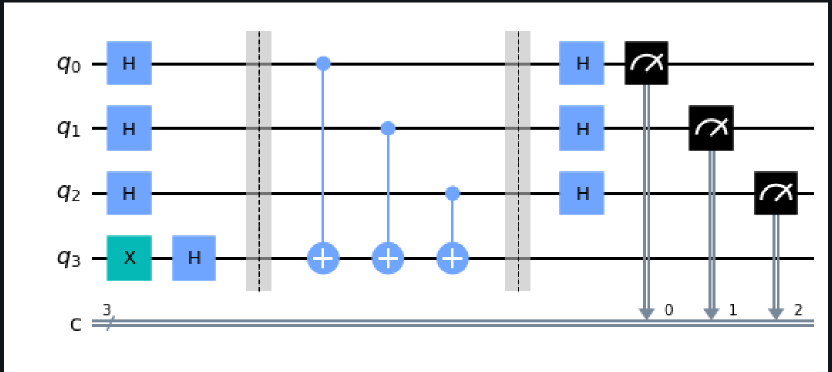
```
SamplerResult(quasi_dists=[{7: 1.0}, {0: 1.0}], metadata=[{'header_metadata': {}, 'shots': 4000}, {'header_metadata': {},
'shots': 4000}])
```

result.quasi_dists

```
[{7: 1.0}, {0: 1.0}]
```

result.quasi_dists

[{7: 1.0}, {0: 1.0}]

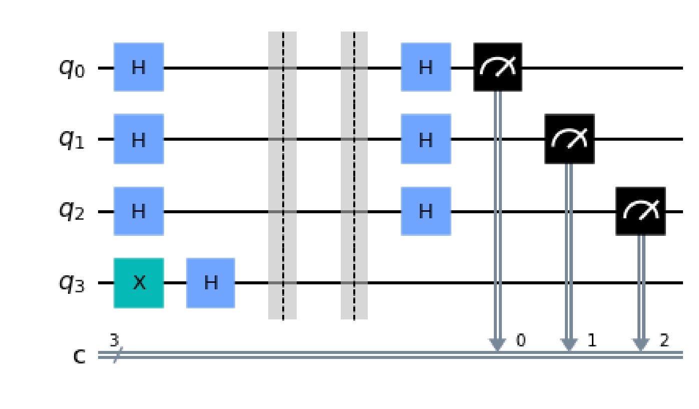


qc1

‘111’: 7 with 1.0 probability

result.quasi_dists

[{7: 1.0}, {0: 1.0}]



qc2

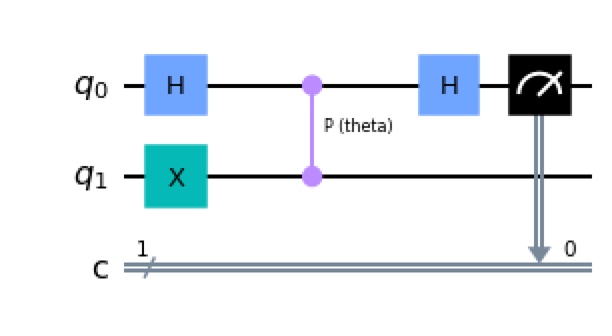
‘000’: 0 with 1.0 probability

Parameterized circuits

Primitive allows simplification of binding multiple parameters in parameterized circuits.

Parameterized circuits

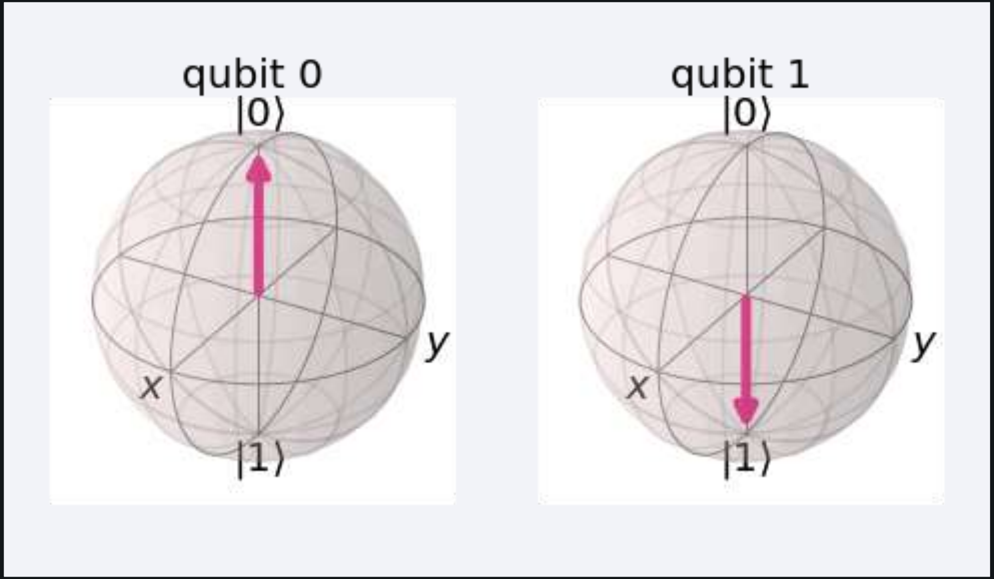
Primitive allows simplification of binding multiple parameters in parameterized circuits.



0 to 2π: divided over 50 evenly spaced points

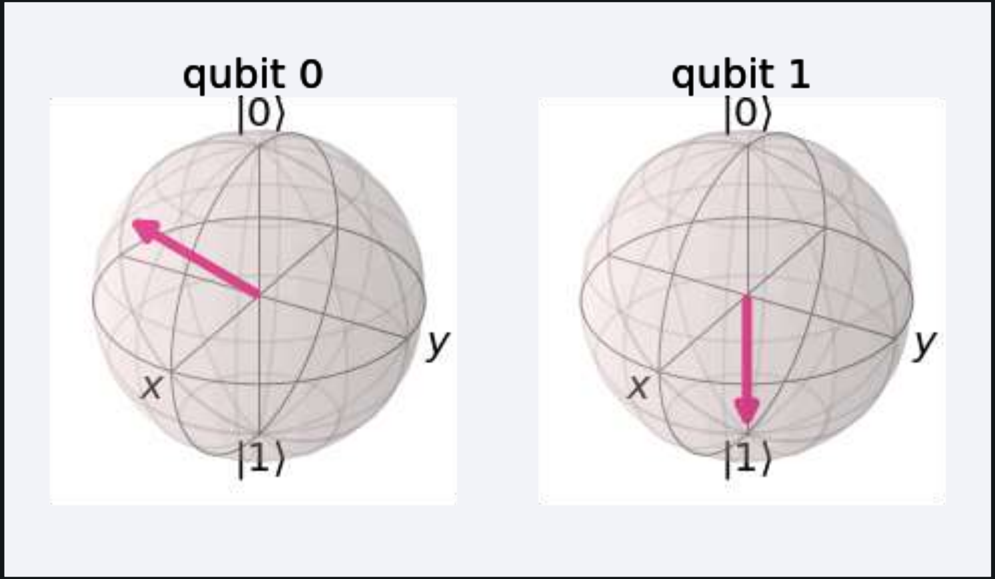
Parameterized circuits

states[0]



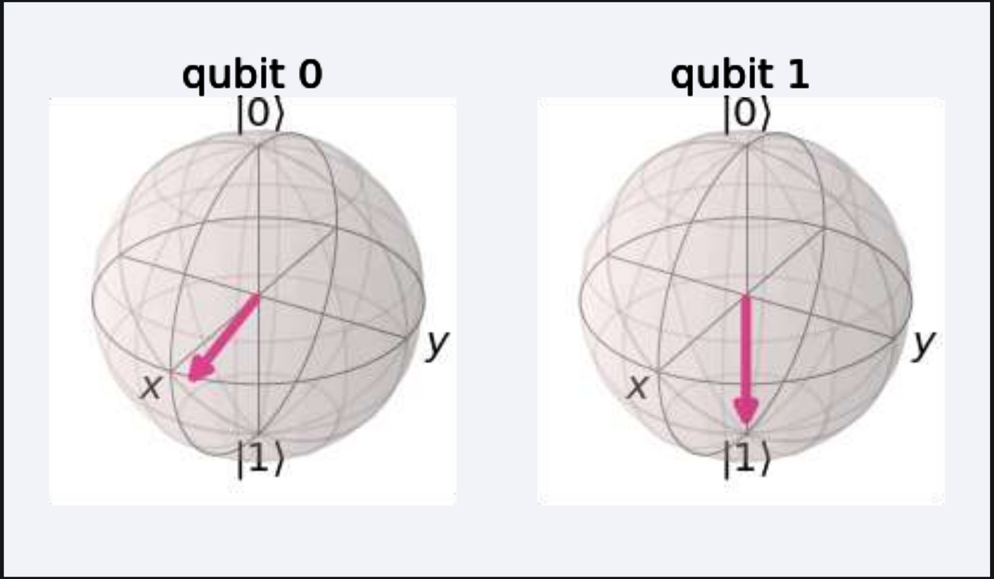
Parameterized circuits

states[1]



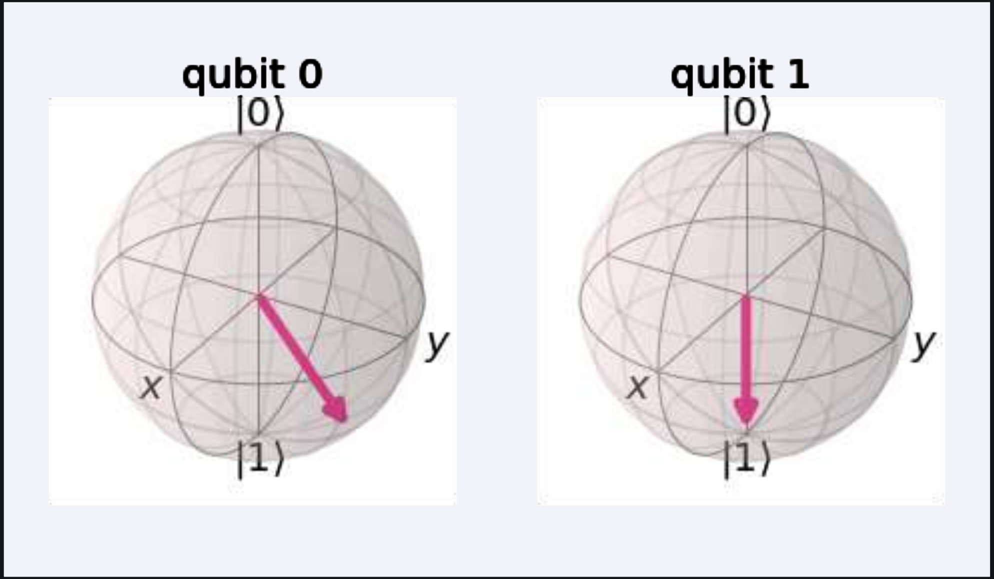
Parameterized circuits

states[2]



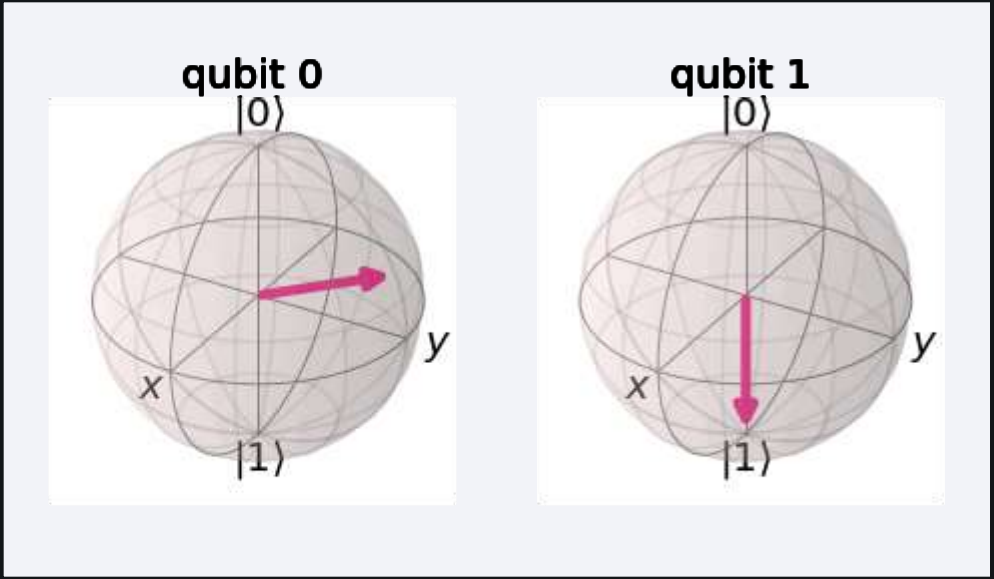
Parameterized circuits

states[3]



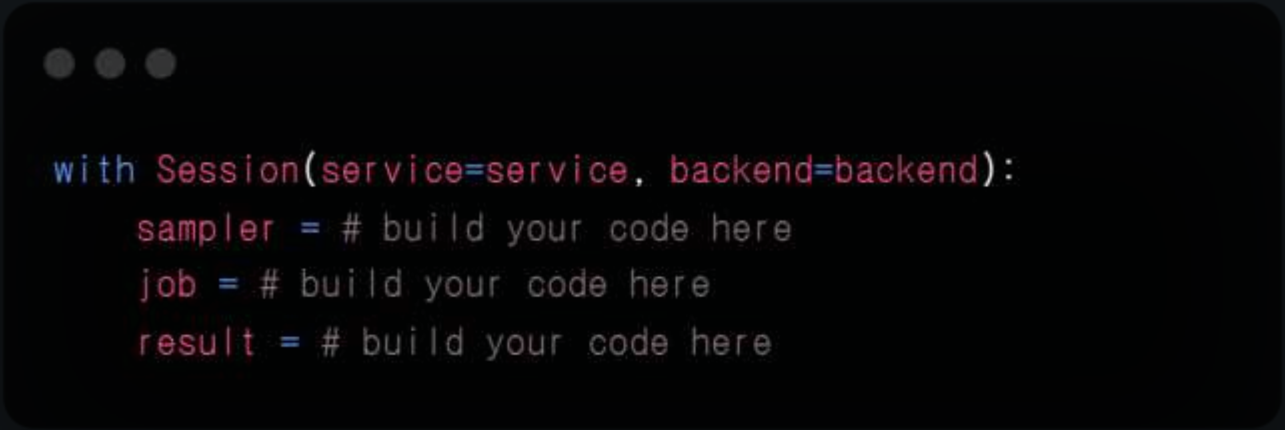
Parameterized circuits

states[4]



Exercise 2

List individual_phases to the qc circuit we made using Sampler



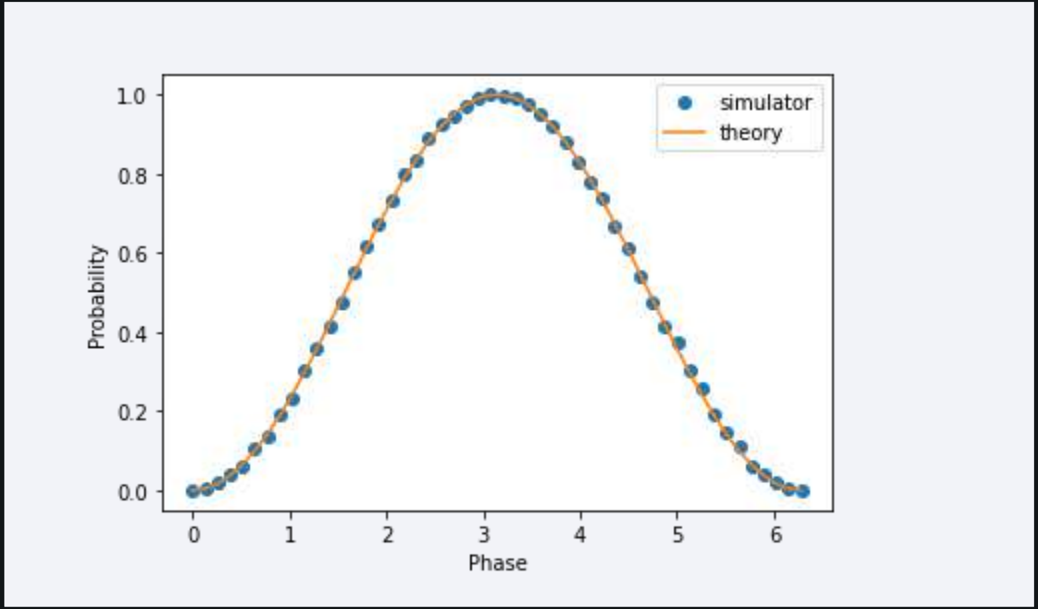
```
with Session(service=service, backend=backend):
    sampler = # build your code here
    job = # build your code here
    result = # build your code here
```

Exercise 2

Parameters for “run” method

- **circuits** - One or more circuit objects
- **parameter_values** - Parameters to be bound to the circuit

Parameterized circuits



Qiskit Runtime Estimator

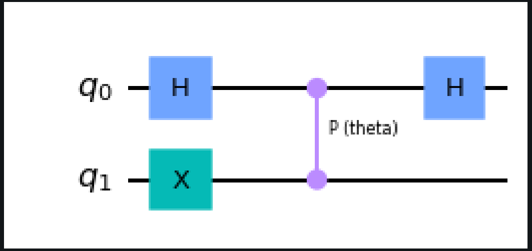
- Efficiently calculate and interpret expectation values of quantum operators required for many algorithms
- Allow selectively group between circuits and observables for execution
  - Efficient evaluation of expectation values and variances for given parameter input.

Qiskit Runtime Estimator

- Efficiently calculate and interpret expectation values of quantum operators required for many algorithms
- Allow selectively group between circuits and observables for execution
  - Efficient evaluation of expectation values and variances for given parameter input.

“No Measurements”

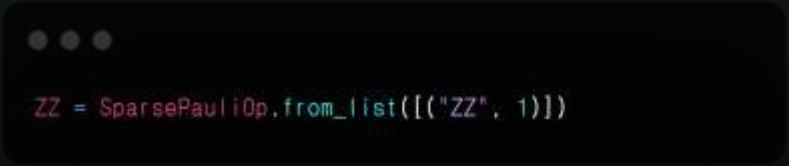
Parametrized Circuit



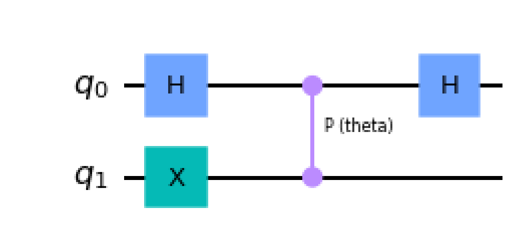
Parametrized Circuit

Calculate expectation value

$$\langle ZZ\rangle = \langle\psi|ZZ|\psi\rangle = \langle\psi|(|0\rangle\langle 0| - |1\rangle\langle 1|) \otimes (|0\rangle\langle 0| - |1\rangle\langle 1|)|\psi\rangle = |\langle 00|\psi\rangle|^2 - |\langle 01|\psi\rangle|^2 - |\langle 10|\psi\rangle|^2 + |\langle 11|\psi\rangle|^2$$

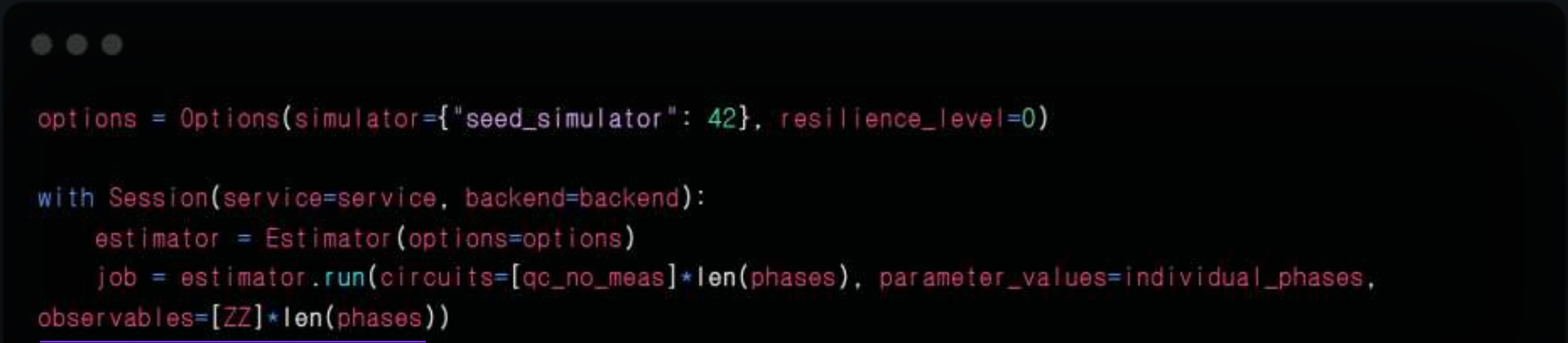


```
ZZ = SparsePauliOp.from_list([("ZZ", 1)])
```

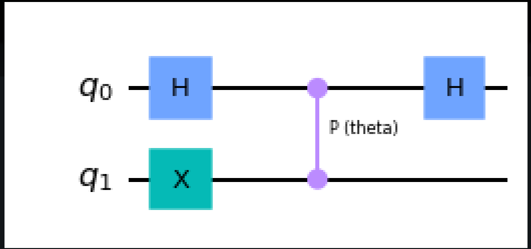


Parametrized Circuit

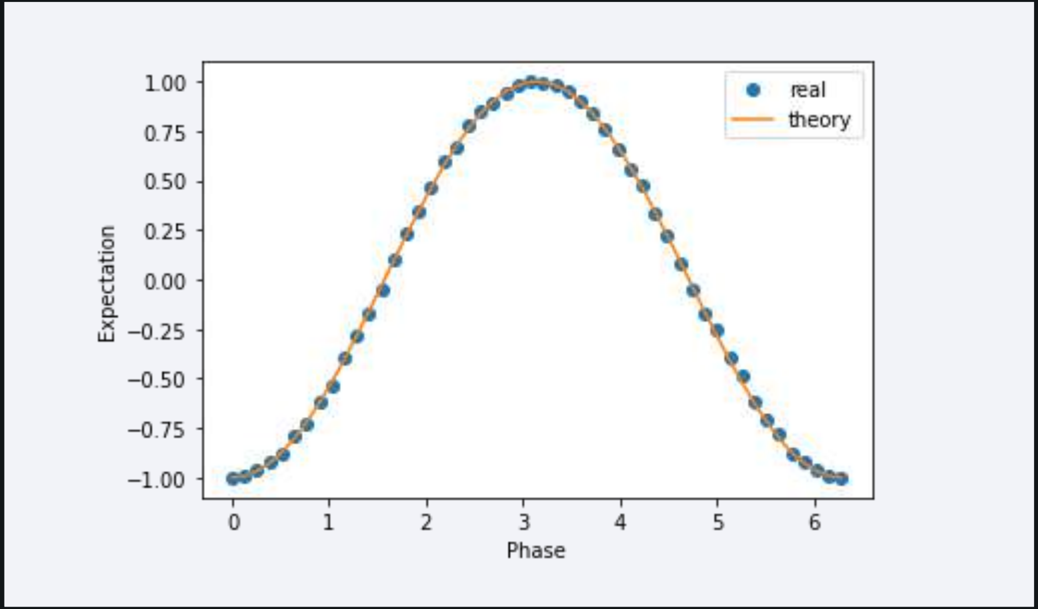
Run using Estimator



```
options = Options(simulator={"seed_simulator": 42}, resilience_level=0)

with Session(service=service, backend=backend):
    estimator = Estimator(options=options)
    job = estimator.run(circuits=[qc_no_meas]*len(phases), parameter_values=individual_phases,
observables=[ZZ]*len(phases))
```



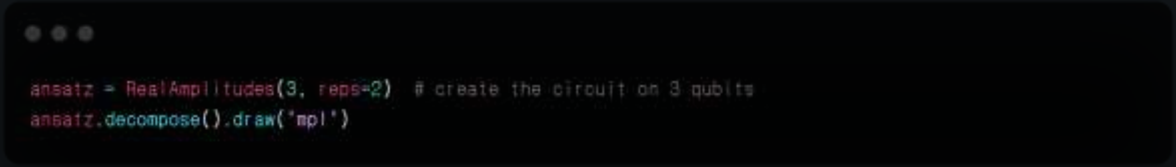
Parametrized Circuit



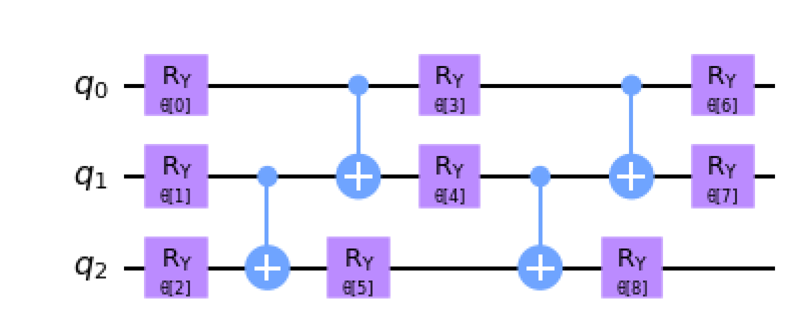
Hamiltonian

- Great candidate for using Estimator
- Quantum mechanical operator
- Has total energy information inside a system including kinetic and potential energy.

Hamiltonian

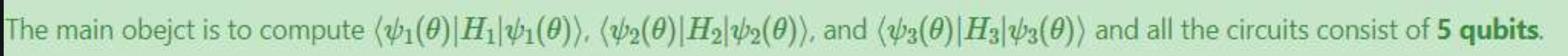


```
ansatz = RealAmplitudes(3, reps=2)  # create the circuit on 3 qubits
ansatz.decompose().draw('mpl')
```



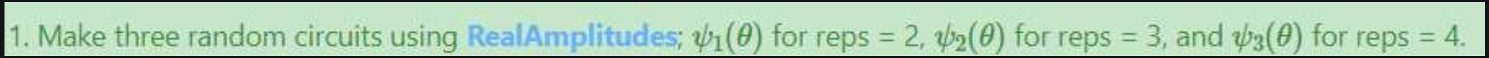
Exercise 3

Build an Estimator routine to compute the expectation values of custom Hamiltonians with respect to certain observables.

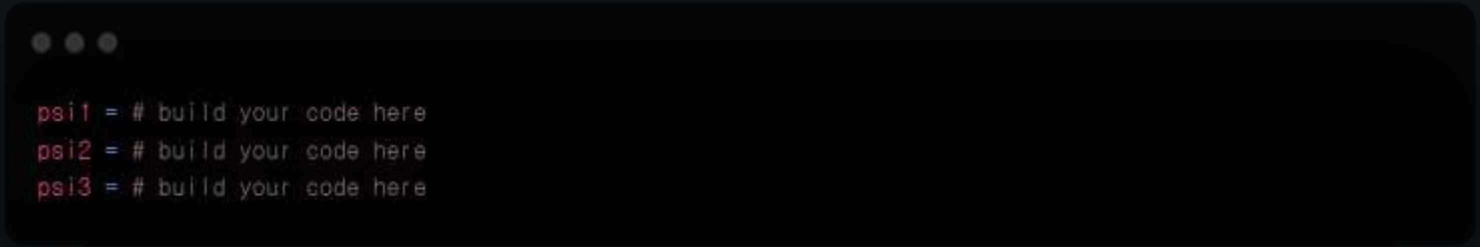


The main obejct is to compute $\langle\psi_1(\theta)|H_1|\psi_1(\theta)\rangle$, $\langle\psi_2(\theta)|H_2|\psi_2(\theta)\rangle$, and $\langle\psi_3(\theta)|H_3|\psi_3(\theta)\rangle$ and all the circuits consist of **5 qubits**.

Exercise 3

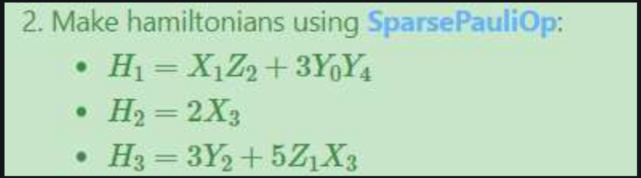


1. Make three random circuits using **RealAmplitudes**; $\psi_1(\theta)$ for reps = 2, $\psi_2(\theta)$ for reps = 3, and $\psi_3(\theta)$ for reps = 4.



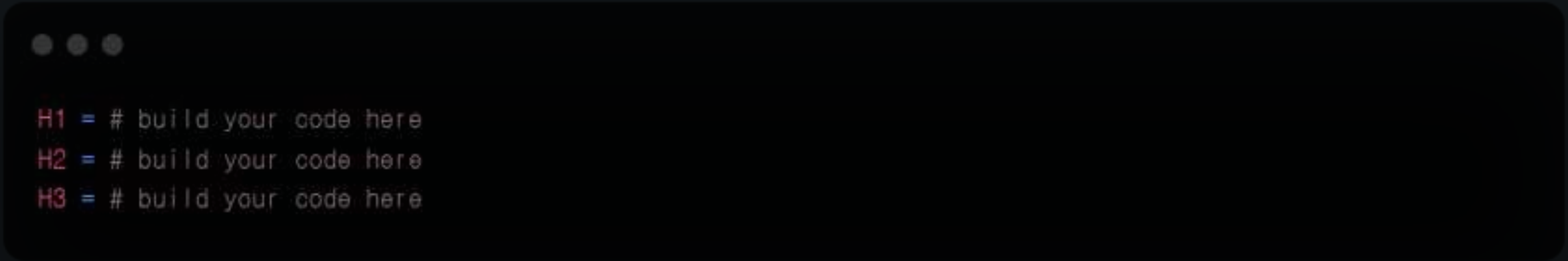
```
psi1 = # build your code here
psi2 = # build your code here
psi3 = # build your code here
```

Exercise 3



2. Make hamiltonians using SparsePauliOp:

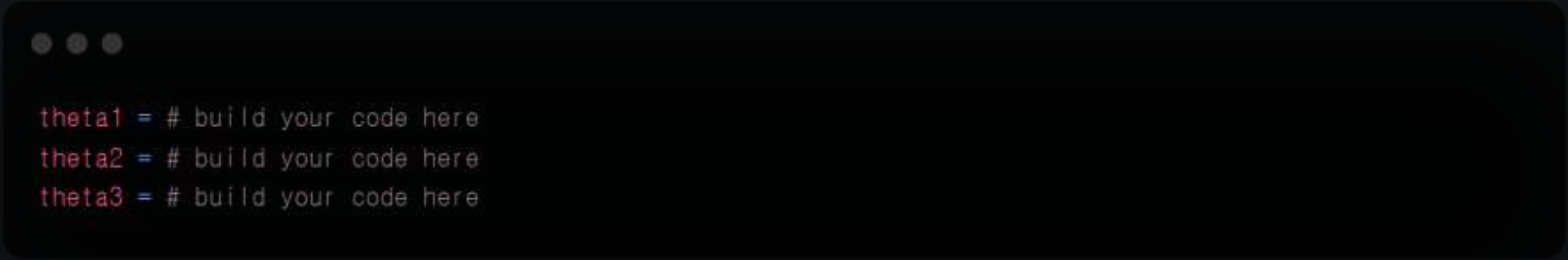
- $H_1 = X_1 Z_2 + 3Y_0 Y_4$
- $H_2 = 2X_3$
- $H_3 = 3Y_2 + 5Z_1 X_3$



```
H1 = # build your code here
H2 = # build your code here
H3 = # build your code here
```

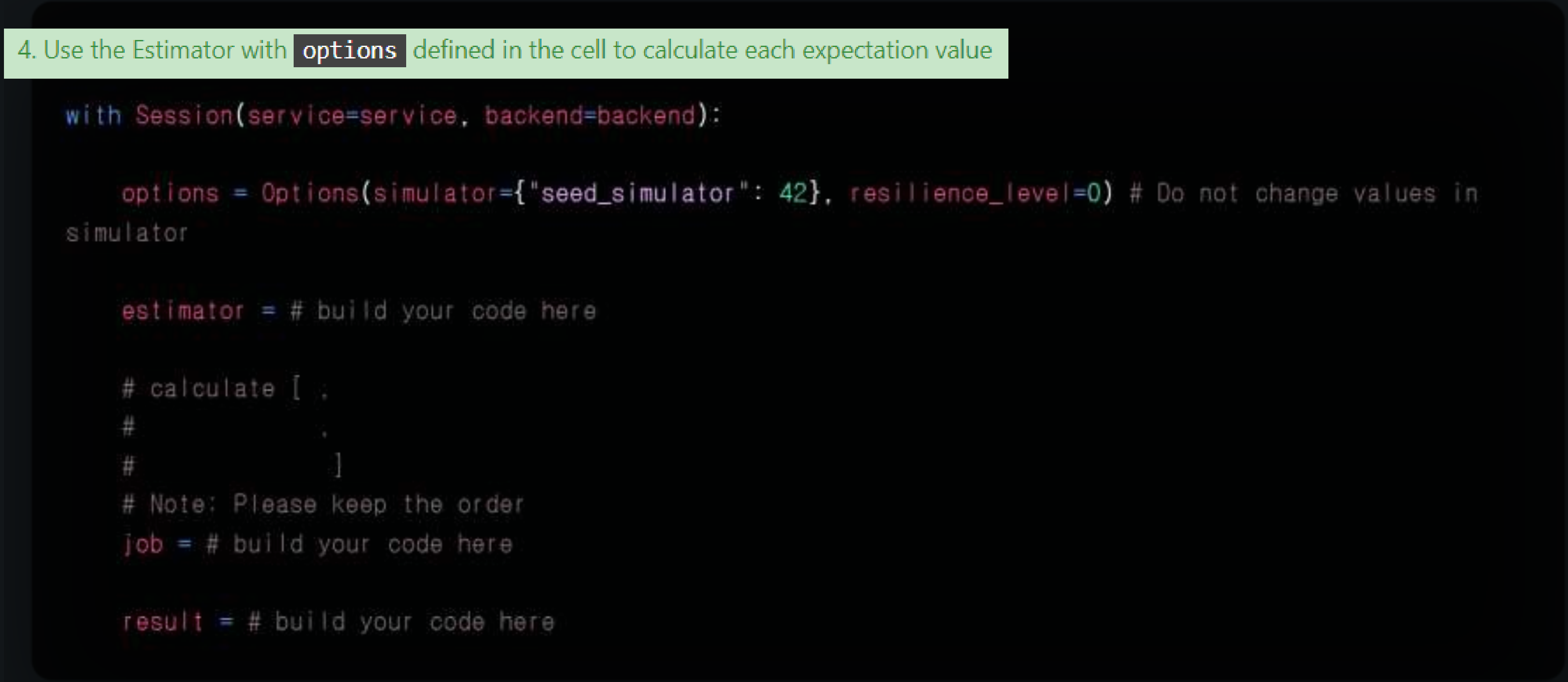
Exercise 3

3. Make a list of evenly spaced values for theta between 0 and 1 using **numpy.linspace**. Note that the number of parameters is different for `reps` of each circuit.



```
theta1 = # build your code here
theta2 = # build your code here
theta3 = # build your code here
```

Exercise 3



4. Use the Estimator with `options` defined in the cell to calculate each expectation value

```
with Session(service=service, backend=backend):

    options = Options(simulator={"seed_simulator": 42}, resilience_level=0) # Do not change values in
simulator

    estimator = # build your code here

    # calculate [ ,
    #            ,
    #             ]
    # Note: Please keep the order
    job = # build your code here

    result = # build your code here
```

Error Mitigation

- **Reduce** the error effects with a much smaller overhead
  - Instead of completely eliminating using a huge amount of resourecs
- Error correction is not practically feasible in the current NISQ
  - Number of qubits
  - Controlled error rates
  - Extra circuit depth and measurements

Error Mitigation with Sampler

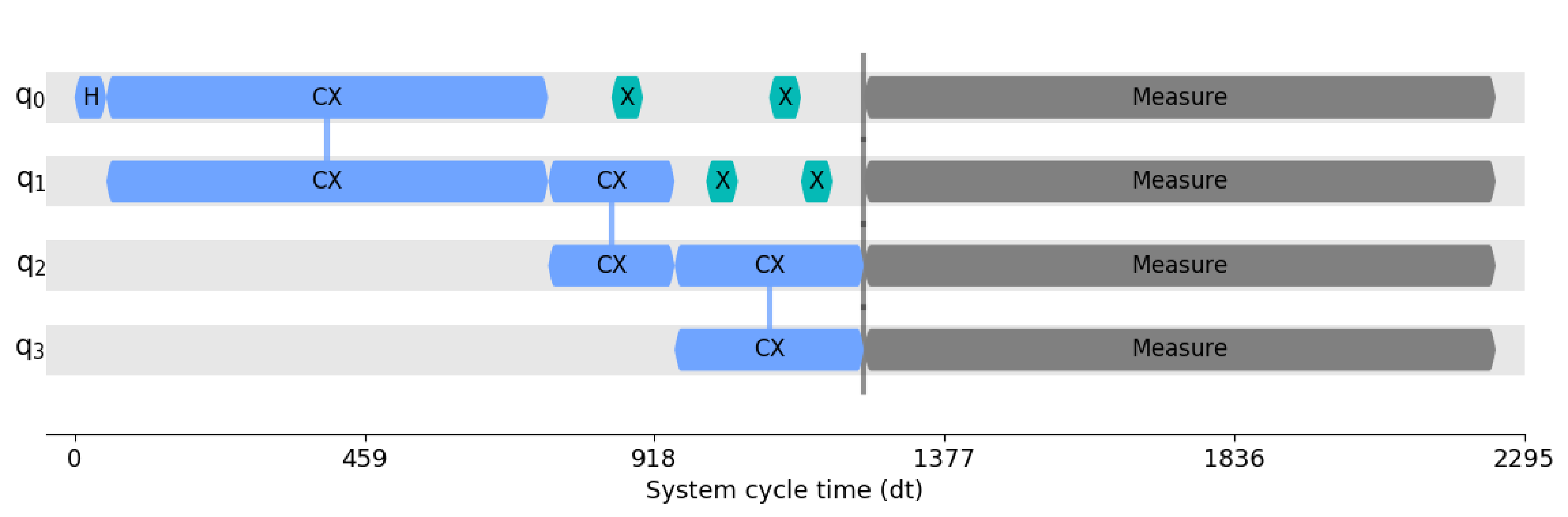
- Dynamic Decoupling (DD)
- Matrix-free Measurement Mitigation (M3)

DD

- This feature is enabled by default on Qiskit Runtime
- Used to increase the lifetime of quantum information

DD

- Effectively disconnect the environment using decoupling methods
  - Prevent idle state of qubit – leakage of information to surrounding due to decoherence.

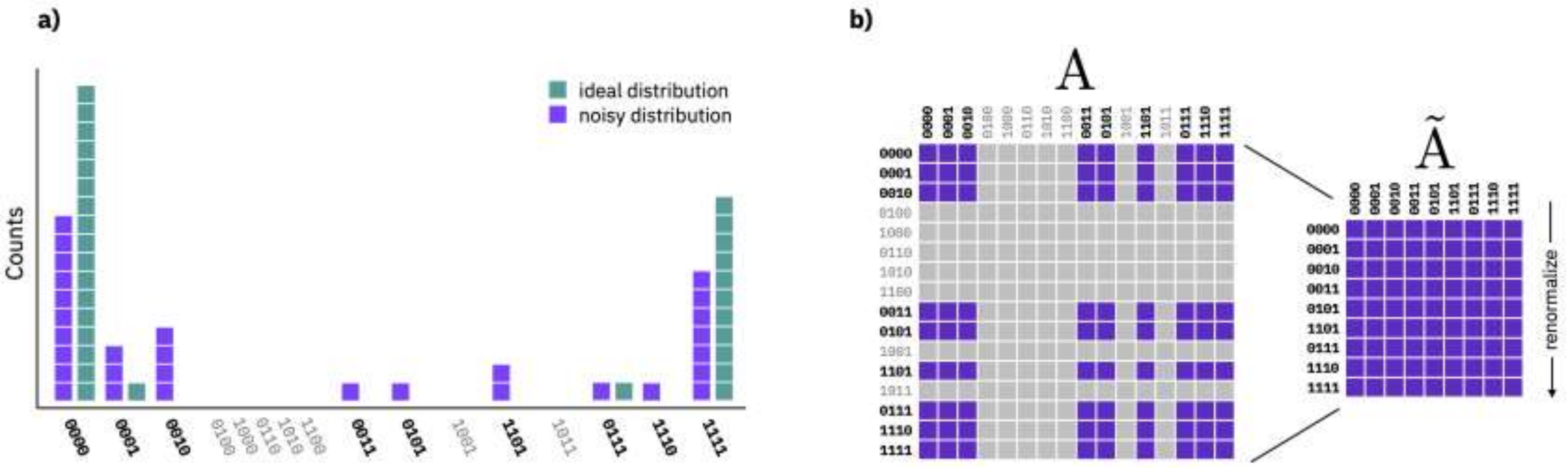


M3

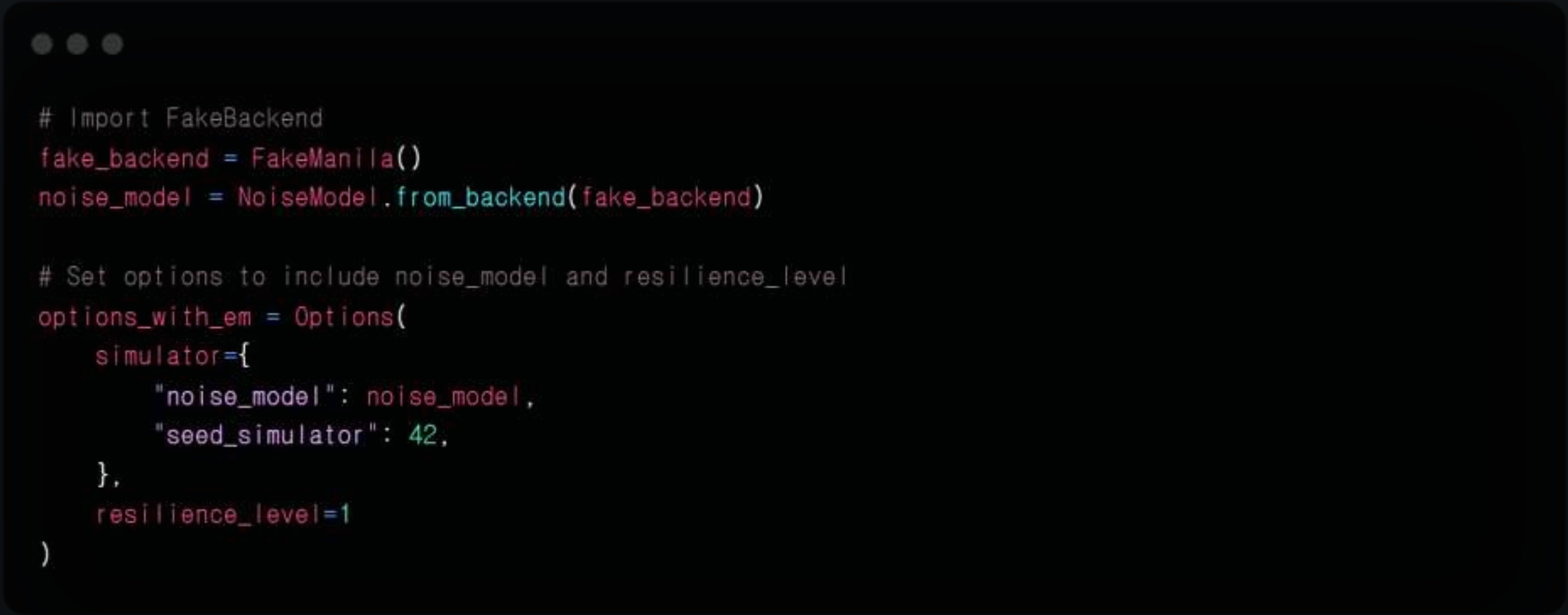
- This feature is enabled by setting `resilience_level=1` in `Options` with Sampler
- Scalable quantum measurement error mitigation
  - Need not explicitly form the assignment matrix or its inverse.

M3

- M3 works in a reduced subspace defined by the noisy input bitstrings that are to be corrected.
- Resulting linear system of equations is nominally much easier to solve.
  - Much smaller number of unique bitstrings than full multi-qubit Hilbert space

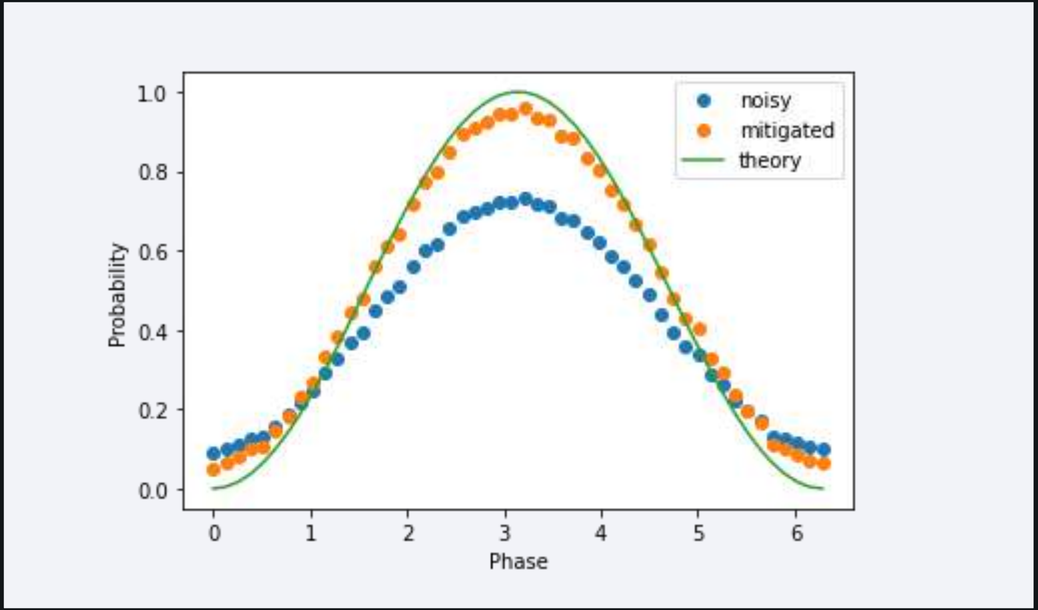


M3



```
# Import FakeBackend
fake_backend = FakeManila()
noise_model = NoiseModel.from_backend(fake_backend)

# Set options to include noise_model and resilience_level
options_with_em = Options(
    simulator={
        "noise_model": noise_model,
        "seed_simulator": 42,
    },
    resilience_level=1
)
```

M3



Probability
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
Phase
● noisy
● mitigated
— theory

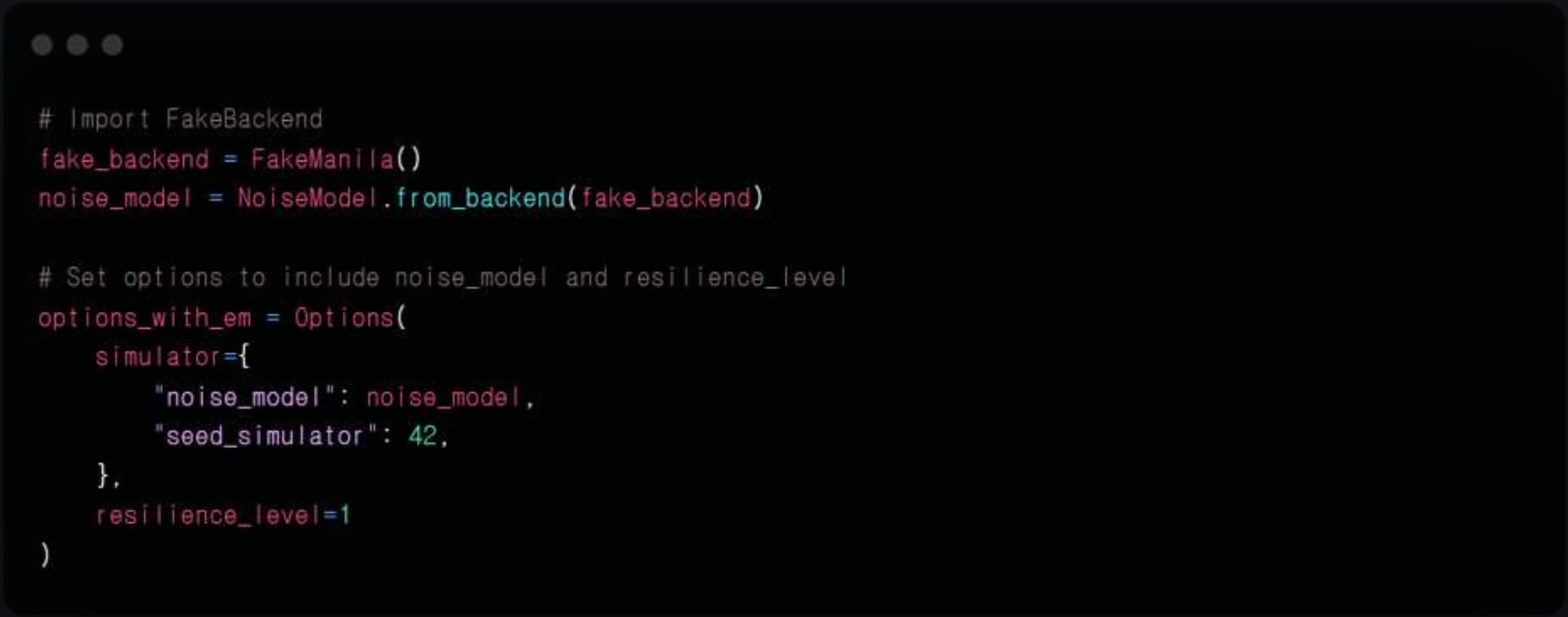
Error Mitigation with Estimator

- Twirled Readout Error eXtinction (T-Rex)
- Zero Noise Extrapolation (Digital ZNE)

T-Rex

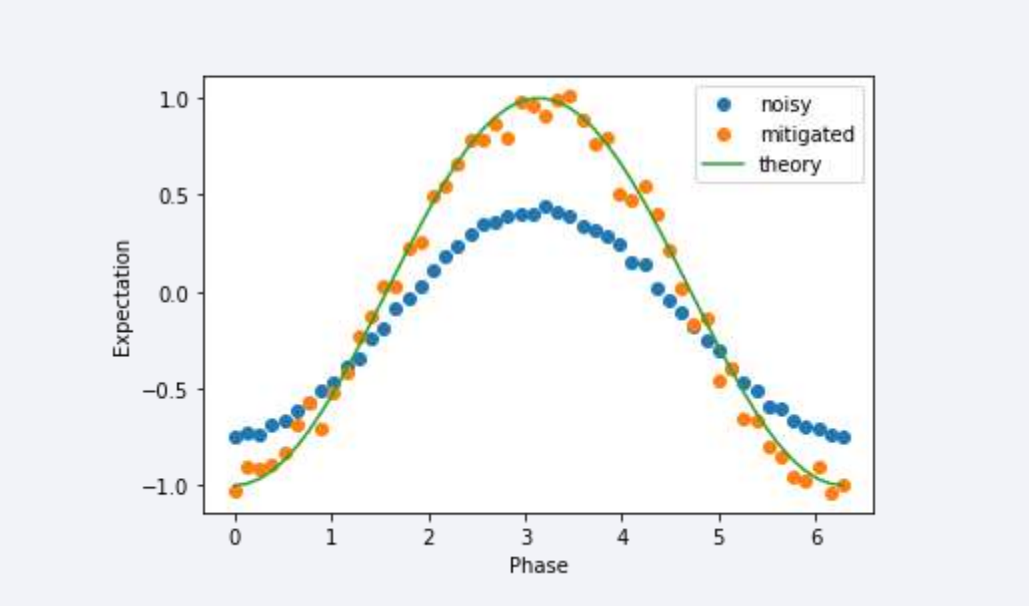
- This feature is enabled by setting `resilience_level=1` in `Options` with Estimator (Default)
- Implementation which involves "twirling" of gates
- View noise as a set of **extra probabilistic gates** on top of our perfect circuit implementation
  - Conjugate this noisy gate set with a gate randomly chosen from a set of gates
- Inserts pairs of Pauli gates (I, X, Y, Z) **before** and **after** entangling gates such that the overall unitary is the same
- Turning **coherent errors** into **stochastic errors**
  - **Stochastic errors can be eliminated by sufficient averaging**

T-Rex



```
# Import FakeBackend
fake_backend = FakeManila()
noise_model = NoiseModel.from_backend(fake_backend)

# Set options to include noise_model and resilience_level
options_with_em = Options(
    simulator={
        "noise_model": noise_model,
        "seed_simulator": 42,
    },
    resilience_level=1
)
```

T-Rex


1.0
0.5
0.0
−0.5
−1.0
Expectation
0
1
2
3
4
5
6
Phase
noisy
mitigated
theory

Digital ZNE

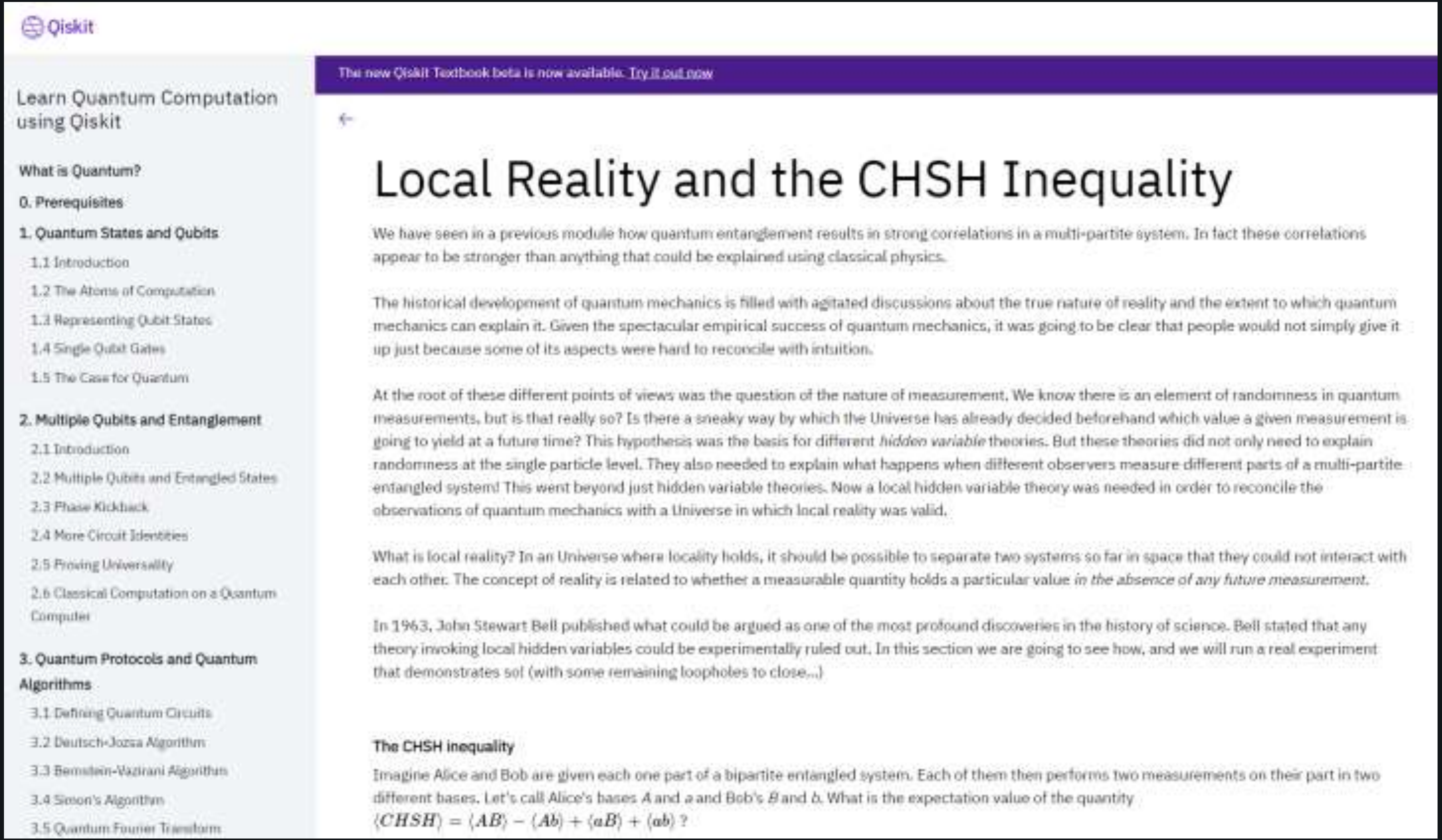
- This feature is enabled by setting `resilience_level=2` in `Options` with Estimator
- Mitigating errors in noisy quantum computers without the need for additional quantum resources.
- A quantum program is altered to run at different effect levels of processor noise
- The result of the computation is extrapolated to an estimated value at a noiseless level.
- Digital way means not using physical pulse.
- Still an active research question around which method is the best to use

Exercise 4

Qiskit

Multiple Choice Question!!!

CHSH inequality



The new Qiskit Textbook beta is now available. Try it out now

Learn Quantum Computation using Qiskit

Local Reality and the CHSH Inequality

We have seen in a previous module how quantum entanglement results in strong correlations in a multi-partite system. In fact these correlations appear to be stronger than anything that could be explained using classical physics.

The historical development of quantum mechanics is filled with agitated discussions about the true nature of reality and the extent to which quantum mechanics can explain it. Given the spectacular empirical success of quantum mechanics, it was going to be clear that people would not simply give it up just because some of its aspects were hard to reconcile with intuition.

At the root of these different points of views was the question of the nature of measurement. We know there is an element of randomness in quantum measurements, but is that really so? Is there a sneaky way by which the Universe has already decided beforehand which value a given measurement is going to yield at a future time? This hypothesis was the basis for different *hidden variable* theories. But these theories did not only need to explain randomness at the single particle level. They also needed to explain what happens when different observers measure different parts of a multi-partite entangled system! This went beyond just hidden variable theories. Now a local hidden variable theory was needed in order to reconcile the observations of quantum mechanics with a Universe in which local reality was valid.

What is local reality? In an Universe where locality holds, it should be possible to separate two systems so far in space that they could not interact with each other. The concept of reality is related to whether a measurable quantity holds a particular value *in the absence of any future measurement.*

In 1963, John Stewart Bell published what could be argued as one of the most profound discoveries in the history of science. Bell stated that any theory invoking local hidden variables could be experimentally ruled out. In this section we are going to see how, and we will run a real experiment that demonstrates so! (with some remaining loopholes to close...)

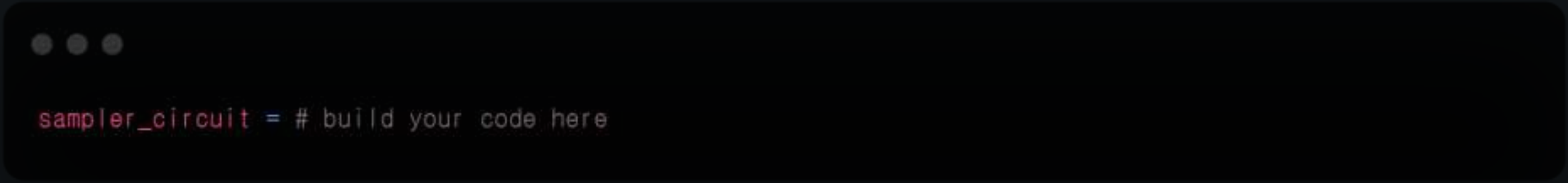
The CHSH inequality

Imagine Alice and Bob are given each one part of a bipartite entangled system. Each of them then performs two measurements on their part in two different bases. Let's call Alice's bases A and a and Bob's B and b. What is the expectation value of the quantity $\langle CHSH \rangle = \langle AB \rangle - \langle Ab \rangle + \langle aB \rangle + \langle ab \rangle$?

Exercise 5

Let's apply error mitigation to the noisy transmission to figure out the hidden message!

1. For SamplerResult, you make a Bernstein-Vazirani circuit with the hidden bitstring, "11111" to execute.

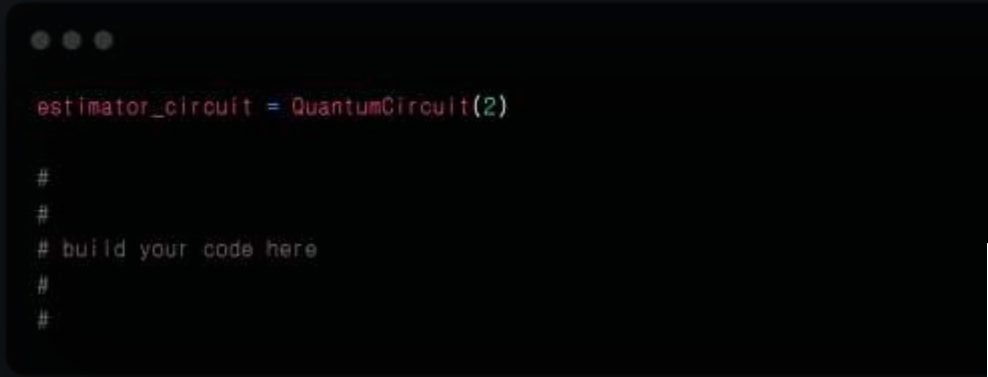


```
sampler_circuit = # build your code here
```

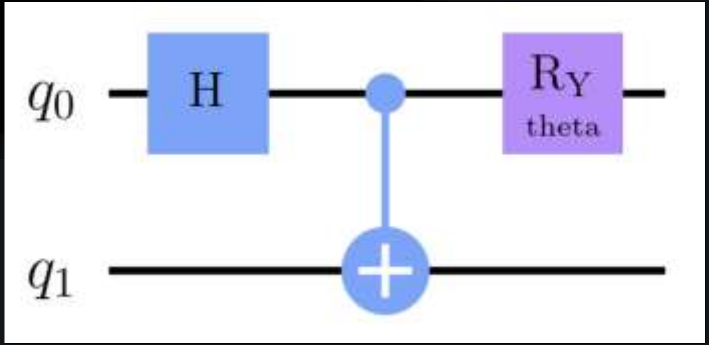
Exercise 5

Let's apply error mitigation to the noisy transmission to figure out the hidden message!

2. For EstimatorResult, build a circuit for S_1 to demonstrate the violation of the CHSH inequality.



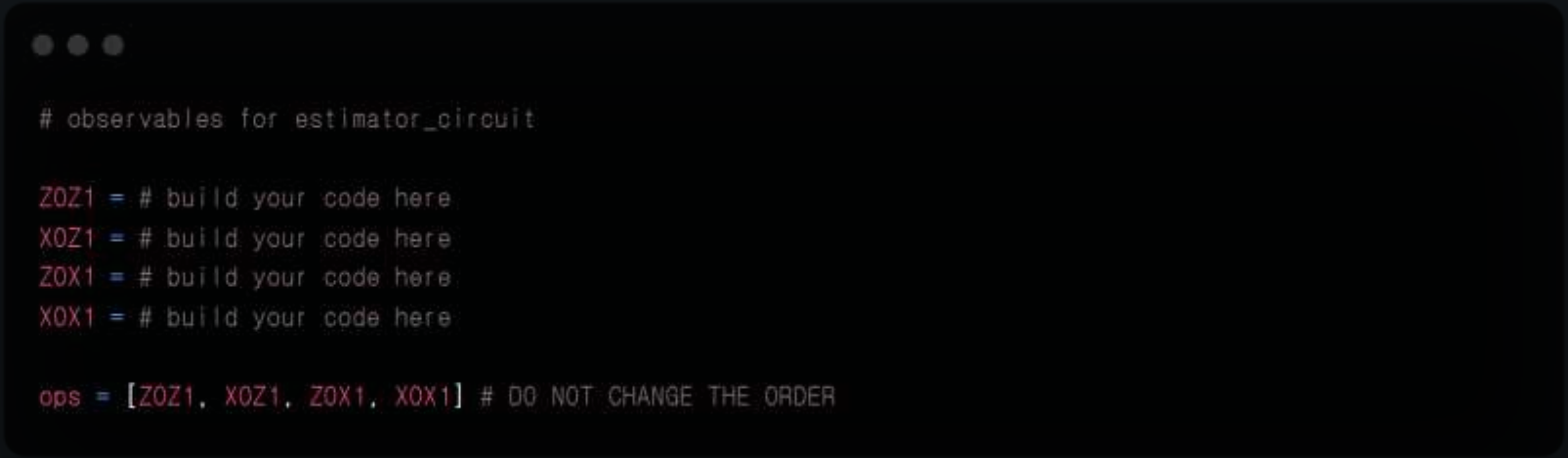
```
estimator_circuit = QuantumCircuit(2)

#
#
# build your code here
#
#
```



Exercise 5

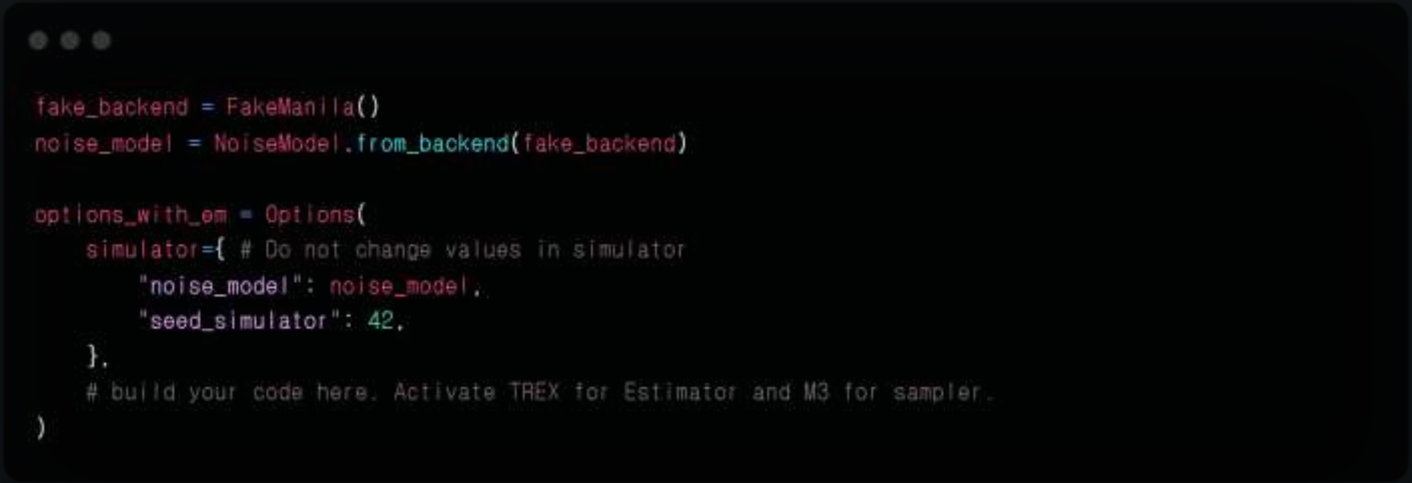
Let's apply error mitigation to the noisy transmission to figure out the hidden message!

Four different EstimatorResult objects: $Z_0Z_1, X_0Z_1, Z_0X_1, X_0X_1$



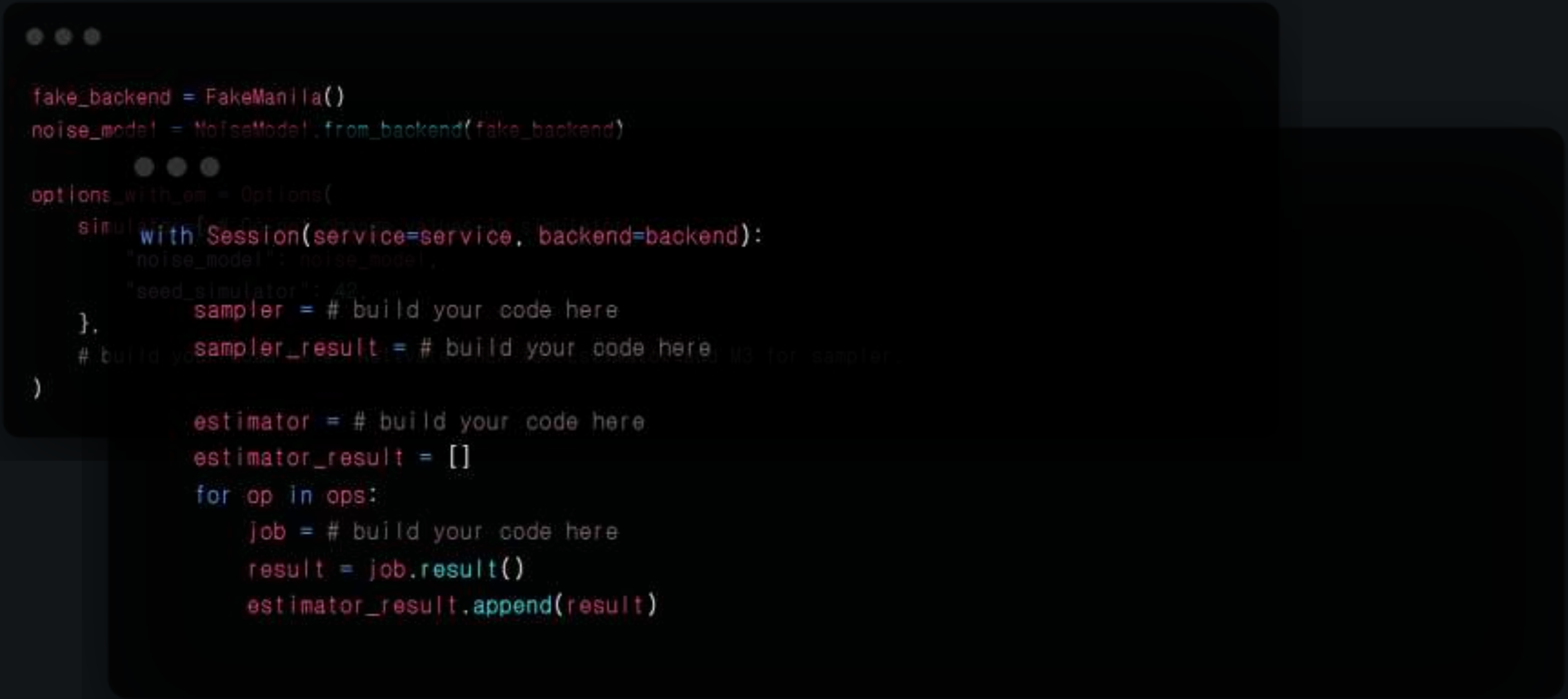
```
# observables for estimator_circuit

Z0Z1 = # build your code here
X0Z1 = # build your code here
Z0X1 = # build your code here
X0X1 = # build your code here

ops = [Z0Z1, X0Z1, Z0X1, X0X1] # DO NOT CHANGE THE ORDER
```

Exercise 5



```
fake_backend = FakeManila()
noise_model = NoiseModel.from_backend(fake_backend)

options_with_em = Options(
    simulator={ # Do not change values in simulator
        "noise_model": noise_model,
        "seed_simulator": 42,
    },
    # build your code here. Activate TREX for Estimator and M3 for sampler.
)
```

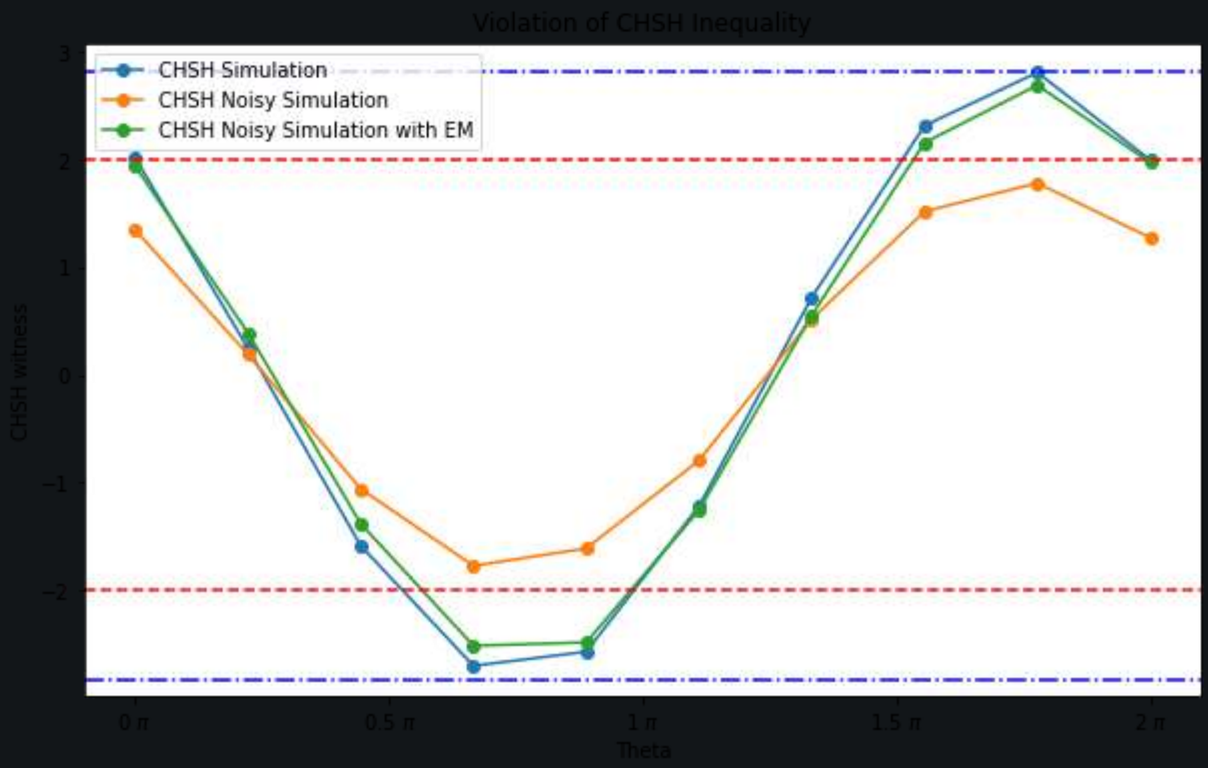
Exercise 5



```
fake_backend = FakeManila()
noise_model = NoiseModel.from_backend(fake_backend)

options_with_em = Options(
    simulator={ [illegible]
        "noise_model": noise_model,
        "seed_simulator": 42,
    },
    # build [illegible] M3 for sampler.
)
```

```
with Session(service=service, backend=backend):

    sampler = # build your code here
    sampler_result = # build your code here

    estimator = # build your code here
    estimator_result = []
    for op in ops:
        job = # build your code here
        result = job.result()
        estimator_result.append(result)
```

Exercise 5


Violation of CHSH Inequality
CHSH witness
Theta
3
2
1
0
−1
−2
0 π
0.5 π
1 π
1.5 π
2 π
CHSH Simulation
CHSH Noisy Simulation
CHSH Noisy Simulation with EM

Exercise 6

Hidden Message!!