Practical 7

Implement hello world using quantum computer

Introduction to Quantum Computing

Quantum computing is an emerging field of computer science that leverages the principles of quantum mechanics to process information. Unlike classical computers that use bits (0 or 1) to represent data, quantum computers use qubits, which can exist in superposition—being both 0 and 1 at the same time. This property, along with entanglement and quantum interference, enables quantum computers to solve certain problems more efficiently than classical ones.

Traditionally, the first step in learning a new programming language or paradigm is to create a simple "Hello World" program. While this concept is straightforward in classical computing, in the quantum realm, it serves a different purpose. Due to the unique nature of quantum computers, a "Hello World" program here is not about printing text, but rather demonstrating the basic functionality of a quantum system, such as qubit initialization, applying quantum gates, measuring qubits, and observing outcomes.

Conceptualizing "Hello World" in Quantum Terms

Since quantum computers are not built to manipulate strings and characters directly like classical systems, implementing a "Hello World" equivalent is more symbolic. The focus is on creating a basic quantum circuit and verifying that it behaves as expected. This serves as an introduction to quantum gates, measurements, and probabilistic outcomes.

To conceptualize a "Hello World" in quantum computing, the following elements are considered:

1. **Qubit Initialization**
   All qubits are initialized in the $|0\rangle$ state, which is analogous to setting classical bits to 0. This state is the starting point for all quantum operations.
2. **Quantum Gates**
   Applying quantum gates is like applying functions or operations to the qubits. For example:
   - **The Hadamard gate (H) places a qubit into a superposition state.**
   - **The Pauli-X gate (X) flips the qubit from $|0\rangle$ to $|1\rangle$ or vice versa.**

- **Multi-qubit gates like CNOT (Controlled-NOT) are used to entangle qubits.**

3. **Measurement**
   Measurement collapses a qubit from a superposition into a definite classical state (0 or 1). This is how we extract information from a quantum system. A simple "Hello World" quantum program may measure the outcome of one or more qubits and display the result in binary form.

Symbolic "Hello World" Output

Since quantum computers don't directly print "Hello World" in a literal sense, the symbolic equivalent is the correct execution and interpretation of measured outputs. For example, if a quantum circuit is set up so that after measurement the qubits consistently output the binary pattern corresponding to the ASCII code of "H" (which is 01001000), that could be considered a symbolic form of "H".

Alternatively, if a series of measurements results in patterns that can be interpreted as bits representing ASCII codes of letters in "Hello World", then the program can be said to "output" the message, albeit indirectly.

This could be done by designing a circuit where:

- **Each character of "Hello World" is represented in ASCII.**
- **Each ASCII character is encoded into a specific combination of qubit states.**
- **Measurements convert those quantum states into classical binary outputs.**
- **These outputs are interpreted and mapped to characters.**

This process, though theoretically possible, is not efficient or practical with current quantum systems, especially for longer strings. However, it serves as a meaningful learning tool for understanding qubit manipulation and measurement.

Educational Purpose of a Quantum "Hello World"

The goal of a quantum "Hello World" is to:

- **Demonstrate the creation and execution of a simple quantum circuit.**
- **Understand basic quantum operations such as superposition, entanglement, and measurement.**
- **Build foundational knowledge for more complex quantum algorithms.**

For example, a basic quantum “Hello World” might:

- **Initialize 1 or 2 qubits.**
- **Apply Hadamard gates to create superposition.**
- **Use CNOT gates to demonstrate entanglement.**
- **Measure the qubits and observe probabilistic outputs (e.g., 00, 01, 10, 11).**

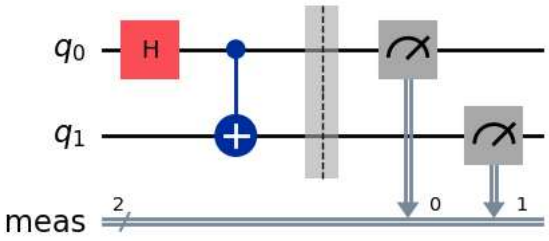
Observing that the output is consistent with theoretical predictions confirms that the quantum circuit behaves correctly—analogous to seeing “Hello World” printed correctly in a classical program.

```
!pip install qiskit-ibm-runtime
```

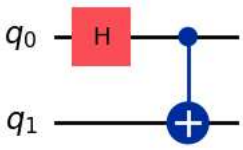
```
Requirement already satisfied: qiskit-ibm-runtime in /usr/local/lib/python3.11/dist-packages (0.38.0)
Requirement already satisfied: requests>=2.19 in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (2.32.3)
Requirement already satisfied: requests-ntlm>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (1.3.0)
Requirement already satisfied: numpy>=1.13 in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (2.0.2)
Requirement already satisfied: urllib3>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (2.3.0)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (2.8.2)
Requirement already satisfied: ibm-platform-services>=0.22.6 in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (0.63.
Requirement already satisfied: pydantic<2.10,>=2.5.0 in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (2.9.2)
Requirement already satisfied: qiskit>=1.4.1 in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (2.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from qiskit-ibm-runtime) (24.2)
Requirement already satisfied: ibm_cloud_sdk_core<4.0.0,>=3.22.1 in /usr/local/lib/python3.11/dist-packages (from ibm-platform-services>
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.10,>=2.5.0->qiskit-ibm
Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.10,>=2.5.0->qiskit-ibm-
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.10,>=2.5.0->qiskit-i
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.0->qiskit-ibm-runtime) (1.
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.4.1->qiskit-ibm-runtime) (0.
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.4.1->qiskit-ibm-runtime) (1.14.1)
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.4.1->qiskit-ibm-runtime) (1.13.1)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.4.1->qiskit-ibm-runtime) (0.4.0)
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.4.1->qiskit-ibm-runtime) (5.4
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.4.1->qiskit-ibm-runtime)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19->qiskit-ibm-runt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19->qiskit-ibm-runtime) (3.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19->qiskit-ibm-runtime) (
Requirement already satisfied: cryptography>=1.3 in /usr/local/lib/python3.11/dist-packages (from requests-ntlm>=1.1.0->qiskit-ibm-runti
Requirement already satisfied: pyspnego>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from requests-ntlm>=1.1.0->qiskit-ibm-runtime
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography>=1.3->requests-ntlm>=1.1.0->qisk
Requirement already satisfied: PyJWT<3.0.0,>=2.8.0 in /usr/local/lib/python3.11/dist-packages (from ibm_cloud_sdk_core<4.0.0,>=3.22.1->i
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from stevedore>=3.0.0->qiskit>=1.4.1->qiskit-ibm-r
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy>=1.3->qiskit>=1.4.1->qiskit-ibm
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography>=1.3->requests-ntlm>=
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from pbr>=2.0.0->stevedore>=3.0.0->qiskit>=1.4.1->
```

```
# Install Qiskit with full support (only needed once)
!pip install -q qiskit[all]
!pip install qiskit qiskit-aer

# Now run the code
from qiskit_aer import Aer
from qiskit.circuit import QuantumCircuit
from qiskit.transpiler import generate_preset_pass_manager
import matplotlib.pyplot as plt

# Step 1: Create your quantum circuit
qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)
qc.measure_all()

# Step 2: Use a local simulator instead of IBM backend
backend = Aer.get_backend('aer_simulator')

# Step 3: Optimize the circuit using the preset pass manager
pm = generate_preset_pass_manager(backend=backend, optimization_level=1)
isa_circuit = pm.run(qc)

# Step 4: Draw the optimized circuit
isa_circuit.draw("mpl", idle_wires=False)
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.11/dist-packages (2.0.0)
Collecting qiskit-aer
  Downloading qiskit_aer-0.17.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.2 kB)
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.11/dist-packages (from qiskit) (0.16.0)
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/python3.11/dist-packages (from qiskit) (2.0.2)
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/dist-packages (from qiskit) (1.14.1)
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/dist-packages (from qiskit) (1.13.1)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/dist-packages (from qiskit) (0.4.0)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.11/dist-packages (from qiskit) (2.8.2)
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.11/dist-packages (from qiskit) (5.4.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from qiskit) (4.13.1)
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/python3.11/dist-packages (from qiskit) (0.13.0)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.11/dist-packages (from qiskit-aer) (5.9.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.0->qiskit) (1.17.0)
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from stevedore>=3.0.0->qiskit) (6.1.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy>=1.3->qiskit) (1.3.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from pbr>=2.0.0->stevedore>=3.0.0->qiskit) (75.2.0
Downloading qiskit_aer-0.17.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 91.3 MB/s eta 0:00:00
Installing collected packages: qiskit-aer
Successfully installed qiskit-aer-0.17.0
```




```
from qiskit import QuantumCircuit
from qiskit.quantum_info import SparsePauliOp
from qiskit.transpiler import generate_preset_pass_manager
from qiskit_ibm_runtime import EstimatorV2 as Estimator

# Create a new circuit with two qubits
qc = QuantumCircuit(2)

# Add a Hadamard gate to qubit 0
qc.h(0)

# Perform a controlled-X gate on qubit 1, controlled by qubit 0
qc.cx(0, 1)

# Return a drawing of the circuit using MatPlotLib ("mpl"). This is the
# last line of the cell, so the drawing appears in the cell output.
# Remove the "mpl" argument to get a text drawing.
qc.draw("mpl")
```




```
# Set up six different observables.

observables_labels = ["IZ", "IX", "ZI", "XI", "ZZ", "XX"]
observables = [SparsePauliOp(label) for label in observables_labels]



# Use the following code instead if you want to run on a simulator:

from qiskit_ibm_runtime.fake_provider import FakeAlmadenV2
backend = FakeAlmadenV2()
estimator = Estimator(backend)

# Convert to an ISA circuit and layout-mapped observables.
```

```
pm = generate_preset_pass_manager(backend=backend, optimization_level=1)
isa_circuit = pm.run(qc)
mapped_observables = [
    observable.apply_layout(isa_circuit.layout) for observable in observable
]

job = estimator.run([(isa_circuit, mapped_observables)])
result = job.result()

# This is the result of the entire submission.  You submitted one Pub,
# so this contains one inner result (and some metadata of its own).

job_result = job.result()

# This is the result from our single pub, which had five observables,
# so contains information on all five.

pub_result = job.result()[0]


# Plot the result

from matplotlib import pyplot as plt

values = pub_result.data.evs

errors = pub_result.data.stds

# plotting graph
plt.plot(observables_labels, values, "-o")
plt.xlabel("Observables")
plt.ylabel("Values")
plt.show()
```