Quantum Blockchain

Pre-Apprenticeship Course

Length: 6-weeks

Competency Level: Beginner

Content Moderator: Infinite 8 Institute, L3C

Lead Instructor: Ean Mikale, JD

Course Description: This six-week course is designed for high school students with limited experience in quantum computing and blockchain technology. The course will provide an introduction to the basics of the InfiNET Quantum Network and its implementation as a Blockchain network. The course will be practical, engaging and purposeful, students will learn the fundamental concepts of quantum computing such as qubits, quantum gates and quantum error correction through hands-on coding exercises using the qiskit library. By the end of the course, students will have a solid understanding of the basics of quantum computing and blockchain technology, they will be able to design and implement simple quantum circuits and understand the potential impact of quantum computing on various industries.

Week 1: Introduction to Quantum Computing and Qubits

- Overview of quantum computing and its potential impact on various industries
- Introduction to qubits and quantum states
- Basic concepts of quantum gates
- Coding Exercise: Use qiskit to create a simple quantum circuit that creates a superposition of states
- Assignment: Research a specific industry that could potentially be impacted by quantum computing and create a presentation on your findings

Week 2: Quantum Gates and Quantum Circuits

- Introduction to quantum gates and their implementation in quantum circuits
- Simulation and testing of quantum circuits using qiskit
- Coding Exercise: Use qiskit to simulate a simple quantum circuit and interpret the results
- Assignment: Design and implement a quantum circuit that solves a specific problem using qiskit

Week 3: Quantum Error Correction and Fault Tolerance

- Introduction to quantum error correction and fault tolerance
- Implementation of error correction techniques in quantum circuits
- Coding Exercise: Use qiskit to implement error correction in a simple quantum circuit
- Assignment: Research a specific error correction technique and explain its implementation in qiskit

Week 4: InfiNET Quantum Network and Blockchain

- Introduction to the InfiNET Quantum Network and its implementation as a Blockchain network
- Understanding the potential impact of the InfiNET Quantum Network on various industries
- Coding Exercise: Use qiskit to create a simple quantum circuit that can be used in the InfiNET Quantum Network
- Assignment: Research a specific use case for the InfiNET Quantum Network and create a proposal for how it can be used in a specific industry

Week 5: Tokenomics and Smart Contracts

- Overview of tokenomics and its application in quantum computing
- Introduction to smart contracts and their implementation in the InfiNET Quantum Network
- Coding Exercise: Use qiskit to create a simple smart contract that can be used in the InfiNET Quantum Network
- Assignment: Research a specific token and create a report on its tokenomics and potential use cases in the context of quantum computing and blockchain

Week 6: Capstone Project

- Designing and implementing a simple quantum circuit that can be used in the InfiNET Quantum Network
- Creating a smart contract that can be used in the InfiNET Quantum Network
- Coding Exercise: Use qiskit to create a simple quantum circuit and smart contract that can be used in the InfiNET Quantum Network
- Assignment: Create a final project that showcases the application of the InfiNET Quantum Network and blockchain technology in a specific industry.

Learning Objectives:

- Understand the basic concepts of quantum computing such as qubits, quantum gates, and quantum error correction
- Learn how to code simulated quantum circuits using the qiskit library
- Understand the potential impact of quantum computing on various industries
- Understand the basics of the InfiNET Quantum Network and its implementation as a blockchain network
- Learn about tokenomics and smart contracts in the context of quantum computing and blockchain
- Be able to design and implement simple quantum circuits and smart contracts that can be used in the InfiNET Quantum Network.

Coding Exercises:

Week 1:

- Use qiskit to create a simple quantum circuit that creates a superposition of states

Week 2:

- Use qiskit to simulate a simple quantum circuit and interpret the results
- Design and implement a quantum circuit that solves a specific problem using qiskit

Week 3:

- Use qiskit to implement error correction in a simple quantum circuit

Week 4:

- Use qiskit to create a simple quantum circuit that can be used in the InfiNET Quantum Network

Week 5:

- Use qiskit to create a simple smart contract that can be used in the InfiNET Quantum Network

#Week 1: The script uses qiskit library to create a quantum circuit with one qubit, then it applies #a Hadamard gate to the qubit, which creates a superposition of states. Finally, it simulates the #circuit and prints the resulting statevector.

```
from qiskit import QuantumCircuit, QuantumRegister, execute, Aer

# Create a quantum register with one qubit
q = QuantumRegister(1)

# Create a quantum circuit
qc = QuantumCircuit(q)

# Add a Hadamard gate to create a superposition of states
qc.h(q[0])

# Execute the circuit on a simulator
backend = Aer.get_backend('statevector_simulator')
job = execute(qc, backend)
result = job.result()

# Print the resulting statevector
print(result.get_statevector(qc))
```

#Week 2: The script uses qiskit library to create a quantum circuit with two qubits and two #classical bits, then it applies a CNOT gate to the two qubits, performs measurements on them, #and simulates the circuit. Finally, it prints the results of the measurements.

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute, Aer

# Create a quantum register with two qubits
q = QuantumRegister(2)
# Create a classical register with two bits
c = ClassicalRegister(2)

# Create a quantum circuit
qc = QuantumCircuit(q, c)
```

```
# Add a CNOT gate to the circuit
qc.cx(q[0], q[1])

# Add a measurement gate to the circuit
qc.measure(q, c)

# Execute the circuit on a simulator
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
result = job.result()

# Print the results of the measurement
print(result.get_counts(qc))
```

#Week 3: The script uses qiskit library to create a quantum circuit with one qubit, then it applies #a Hadamard gate and a measurement gate to the qubit and simulates the circuit. It then uses the #CompleteMeasFitter method to perform error correction on the measurement results and prints #the corrected results.

```
from qiskit import QuantumCircuit, QuantumRegister, execute, Aer
from qiskit.ignis.mitigation.measurement import CompleteMeasFitter

# Create a quantum register with one qubit
q = QuantumRegister(1)

# Create a quantum circuit
qc = QuantumCircuit(q)

# Add a Hadamard gate to the circuit
qc.h(q[0])

# Add a measurement gate to the circuit
qc.measure(q, 0)

# Execute the circuit on a simulator
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
result = job.result()

# Perform error correction using CompleteMeasFitter
meas_fitter = CompleteMeasFitter(result, qc)
corrected_results = meas_fitter.filter.apply(result)

# Print the results of the error corrected measurement
print(corrected_results.get_counts(qc))
```

#Week 4: The script uses qiskit library to create a quantum circuit with two qubits, then it #applies a Controlled-Z gate to the two qubits and simulates the circuit. Finally, it prints the #resulting state-vector.

```
from qiskit import QuantumCircuit, QuantumRegister, execute, Aer

# Create a quantum register with two qubits
q = QuantumRegister(2)

# Create a quantum circuit
qc = QuantumCircuit(q)

# Add a Controlled-Z gate to the circuit
qc.cz(q[0], q[1])

# Execute the circuit on a simulator
backend = Aer.get_backend('statevector_simulator')
job = execute(qc, backend)
result = job.result()

# Print the resulting statevector
print(result.get_statevector(qc))
```

#Week 5: The script uses qiskit library to create a quantum circuit with one qubit and one #classical bit, it applies a Hadamard gate and a X gate to the qubit, performs measurements on #them and simulates the circuit. Finally, it prints the results of the measurement.

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute, Aer
from qiskit.circuit.library import HGate, XGate

# Create a quantum register with one qubit
q = QuantumRegister(1)
# Create a classical register with one bit
c = ClassicalRegister(1)

# Create a quantum circuit
qc = QuantumCircuit(q, c)

# Add a Hadamard gate to the circuit
qc.append(HGate(), [q[0]])

# Add a X gate to the circuit
qc.append(XGate(), [q[0]])

# Add a measurement gate to the circuit
qc.measure(q, c)

# Execute the circuit on a simulator
backend = Aer.get_backend('qasm_simulator')
```

```
job = execute(qc, backend, shots=1024)
result = job.result()

# Print the results of the measurement
print(result.get_counts(qc))
```

```Extra Credit on Tensors - This script allows the user to input the number of dimensions they want the tensor to have, and it automatically creates a state tensor of all zeroes with the first element set to 1, and reshapes it to the specified number of dimensions. The script then creates a quantum circuit, measures the execution time, prepares the qubits in the state represented by the tensor, applies a controlled-Z gate to the circuit, evolves the state tensor and sends it across a simulated quantum network. After the circuit is applied, the script calculates the error rate by comparing the resulting tensor with an ideal one, and measures the time it takes to run the circuit. Finally, it prints the execution time, error rate, and the final state tensor.
This script is a good example of how to create a quantum circuit with a specific number of qubits, and how to prepare the qubits in a specific state. It also illustrates how to evolve a tensor using a quantum circuit and how to calculate the error rate and the execution time.```

```
import torch
from pytorch_quantum import qhe
import time

# Get the number of dimensions from the user
dimensions = int(input("Enter the number of dimensions (1, 2, 4, 8): "))

# Create an initial state tensor of all zeroes
state_tensor = torch.zeros(2 ** dimensions)

# Set the first element to 1
state_tensor[0] = 1

# Reshape the tensor to the specified number of dimensions
state_tensor = state_tensor.view(*[2] * dimensions)

# Create a quantum circuit
qc = qhe.QuantumCircuit(2 ** (dimensions // 2))

# Measure the execution time
start_time = time.time()

# Preparing the qubits in the state represented by the tensor
qc.prepare_state(state_tensor)

# Applying a controlled-Z gate to the circuit
qc.cz(0, 1)

# Applying the circuit to the state tensor
state_tensor = qc.evolve(state_tensor)
```

```
# sending the tensor across a simulated quantum network
qc2 = qhe.QuantumCircuit(2 ** (dimensions // 2))
qc2.teleport(0, 1)
state_tensor = qc2.evolve(state_tensor)

# Measure the execution time
end_time = time.time()

# Calculate error rate
error_rate = (torch.norm(state_tensor - state_tensor_ideal) ** 2).item()

print("Execution time: ", end_time - start_time)
print("Error rate: ", error_rate)
print(state_tensor)
```