Roll : BEAD22547

Name : Omkar Thakur

```
!pip install qiskit
!pip install qiskit_aer
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.12/dist-packages (2.1.2)
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.12/dist-packages (from d
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages
Collecting qiskit_aer
  Downloading qiskit_aer-0.17.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.
Requirement already satisfied: qiskit>=1.1.0 in /usr/local/lib/python3.12/dist-packages (fr
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.12/dist-packages (fr
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.12/dist-packages (from d
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from py
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.12/dist-packages (from d
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages
Downloading qiskit_aer-0.17.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 87.2 MB/s eta 0:00:00
Installing collected packages: qiskit_aer
Successfully installed qiskit_aer-0.17.1
```

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit_aer import AerSimulator
import numpy as np
```

```
class QuantumRNG:
    def __init__(self, qubits=16):
        self.qubits = qubits
        self.sim = AerSimulator()

    def create_circuit(self):
        """Step 1-2: Initialize registers and create circuit"""
        qr = QuantumRegister(self.qubits, 'q')
        cr = ClassicalRegister(self.qubits, 'c')
        circuit = QuantumCircuit(qr, cr)
        return circuit, qr, cr

    def add_superposition(self, circuit, qr):
        """Step 3: Apply Hadamard gates for superposition"""
        circuit.h(qr)  # Apply H gate to all qubits at once
        return circuit

    def measure_all(self, circuit, qr, cr):
```

```
        """Step 4: Measure qubits"""
        circuit.measure(qr, cr)
        return circuit

    def generate(self):
        """Generate one random number (0 to 2^qubits - 1)"""
        circuit, qr, cr = self.create_circuit()
        circuit = self.add_superposition(circuit, qr)
        circuit = self.measure_all(circuit, qr, cr)

        result = self.sim.run(circuit, shots=1).result()
        binary = list(result.get_counts().keys())[0]
        return int(binary, 2)
```

```
def random_int(max_val=None):
    """Generate random integer. If max_val given, returns 0 to max_val-1"""
    rng = QuantumRNG()
    num = rng.generate()
    return num % max_val if max_val else num
```

```
def random_float():
    """Generate random float between 0 and 1"""
    return random_int() / (2**16 - 1)

def random_choice(items):
    """Randomly select from list"""
    return items[random_int(len(items))]

def random_password(length=8):
    """Generate random password"""
    chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
    return ''.join([random_choice(chars) for _ in range(length)])
```

```
def demo():
    print("16-Qubit Quantum Random Number Generator")
    print("=" * 40)

    # Basic usage
    print(f"Random number: {random_int()}")
    print(f"Random 1-100: {random_int(100) + 1}")
    print(f"Random float: {random_float():.4f}")
    print(f"Random choice: {random_choice(['A', 'B', 'C', 'D'])}")
    print(f"Random password: {random_password(12)}")

    # Multiple numbers
    print(f"\n10 Random numbers:")
    for i in range(10):
        print(f"  {i+1:2d}. {random_int():5d}")
```

```
def test_circuit():
```

```
    """Show circuit structure"""
    rng = QuantumRNG(4)  # 4-qubit for clean display
    circuit, qr, cr = rng.create_circuit()
    circuit = rng.add_superposition(circuit, qr)
    circuit = rng.measure_all(circuit, qr, cr)

    print("4-Qubit Circuit (16-qubit same pattern):")
    print(circuit.draw(output='text'))
```

```
def test_distribution(n=100):
    """Test randomness distribution"""
    numbers = [random_int(10) for _ in range(n)]
    counts = {i: numbers.count(i) for i in range(10)}

    print(f"\nDistribution test (n={n}, range 0-9):")
    for digit, count in counts.items():
        print(f"  {digit}: {'' * (count * 20 // n)} {count}")
```

```
demo()
test_circuit()
test_distribution()
```

```
16-Qubit Quantum Random Number Generator
========================================
Random number: 27636
Random 1-100: 24
Random float: 0.0304
Random choice: B
Random password: jpvMDY95fp1o

10 Random numbers:
   1. 24217
   2. 35271
   3. 57937
   4. 15813
   5.  8585
   6. 52218
   7. 24923
   8. 19652
   9. 28316
  10.  4636
4-Qubit Circuit (16-qubit same pattern):
     ┌───┐ ┌─┐
q_0: ┤ H ├─┤M├──────────
     ├───┤ └╥┘┌─┐
q_1: ┤ H ├──╫─┤M├───────
     ├───┤  ║ └╥┘┌─┐
q_2: ┤ H ├──╫──╫─┤M├────
     ├───┤  ║  ║ └╥┘┌─┐
q_3: ┤ H ├──╫──╫──╫─┤M├─
     └───┘  ║  ║  ║ └╥┘
c: 4/═══════╩══╩══╩══╩══
            0  1  2  3

Distribution test (n=100, range 0-9):
  0:  10
  1:  8
  2:  9
```

```
3: ▌▌ 12
4: ▌ 8
5: ▌▌ 10
6: ▌▌ 13
7: ▌▌▌ 15
8: ▌ 8
9: ▌ 7
```