```python
import numpy as np
import random
```

```python
def add_noise(data, prob=0.2):
    return [1-bit if random.random() < prob else bit for bit in data]
```

```python
def encode(data):
    G = [[1,0,0,0,1,1,0],[0,1,0,0,1,0,1],[0,0,1,0,0,1,1],[0,0,0,1,1,1,1]]
    return [sum(data[i]*G[i][j] for i in range(4)) % 2 for j in range(7)]
```

```python
def decode(received):
    H = [[1,1,0,1,1,0,0],[1,0,1,1,0,1,0],[0,1,1,1,0,0,1]]
    syndrome = [sum(H[i][j]*received[j] for j in range(7)) % 2 for i in range(3)]
    error_pos = syndrome[0]*4 + syndrome[1]*2 + syndrome[2]

    if error_pos > 0:
        received[error_pos-1] = 1 - received[error_pos-1]
    return received[:4]

# Step 3: Quantum 3-Qubit Code
def quantum_encode(state):
    return state[0] * np.array([1,0,0,0,0,0,0,0]) + state[1] * np.array([0,0,0,0,

def quantum_decode(encoded):
    probs = np.abs(encoded)**2
    bits = format(np.argmax(probs), '03b')
    return np.array([1,0]) if sum(int(b) for b in bits) < 2 else np.array([0,1])

# Step 4: Test Functions
def test_classical():
    data = [1,0,1,1]
    encoded = encode(data)
    noisy = add_noise(encoded)
    corrected = decode(noisy)
    print(f"Original: {data}, Corrected: {corrected}, Success: {data==corrected}"

def test_quantum():
    state = np.array([1,0])  # |0⟩
    encoded = quantum_encode(state)
    decoded = quantum_decode(encoded)
    print(f"Quantum fidelity: {abs(np.dot(state, decoded))**2:.3f}")

# Step 5: Performance Test
def compare_performance(trials=50):
    success_ecc = success_direct = 0

    for _ in range(trials):
        data = [random.randint(0,1) for _ in range(4)]

        # With ECC
        encoded = encode(data)
        noisy = add_noise(encoded, 0.2)
        corrected = decode(noisy)
        if corrected == data: success_ecc += 1

        # Without ECC
        noisy_direct = add_noise(data, 0.2)
        if noisy_direct == data: success_direct += 1

    print(f"Success with ECC: {success_ecc/trials:.0%}")
    print(f"Success without ECC: {success_direct/trials:.0%}")
```

```
print("=== Error Correction Demo ===")
test_classical()
test_quantum()
compare_performance()
print("=== Complete ===")
```

```
=== Error Correction Demo ===
Original: [1, 0, 1, 1], Corrected: [1, 0, 1, 0], Success: False
Quantum fidelity: 1.000
Success with ECC: 38%
Success without ECC: 48%
=== Complete ===
```