A Tutorial on

# Grover’s Algorithm

By Felix Arkle and Sebastian Gotto

Introduction

# What Are Our Goals?

## Student Before the Tutorial

### Understanding

- Limited grasp of core concepts (oracle, amplitude amplification, iteration count)
- Grover's speedup feels abstract

### Skills & Confidence

- Uncertain how to translate theory into code
- Rely on trial-and-error in simulators

### Goal Alignment

- Key learning goals not yet met

## Student After the Tutorial

### Understanding

- Clear mental model of oracle and diffuser
- Can explain why $\sqrt{N}$ iterations suffice

### Skills & Confidence

- Implement each Grover subroutine reliably
- Confident running and interpreting simulations
- Inspired to learn more about Quantum Computing

### Goal Alignment

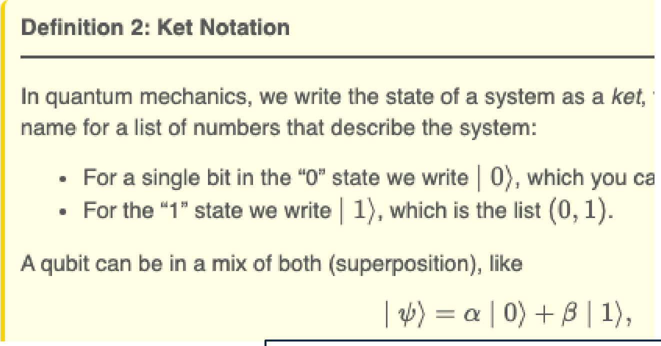
- All learning goals achieved

# How We Aim to Achieve those Goals

1 Intuitive Definitions & Analogies

2 Progressive Challenges

3 Additional Food For Thought



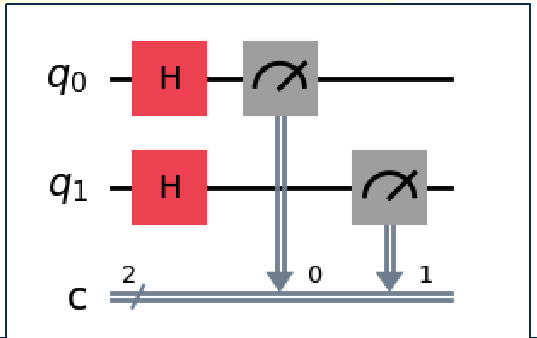
**Definition 2: Ket Notation**

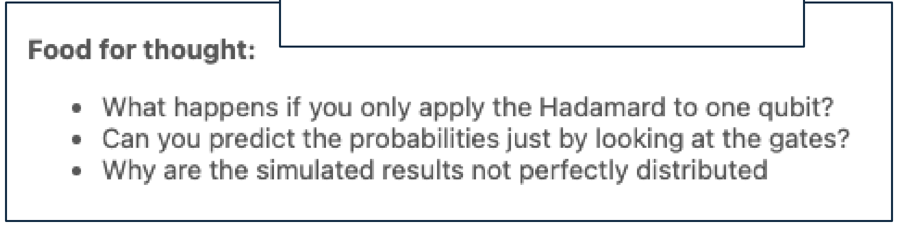
In quantum mechanics, we write the state of a system as a *ket*, '
name for a list of numbers that describe the system:

- For a single bit in the "0" state we write $|\,0\rangle$, which you ca
- For the "1" state we write $|\,1\rangle$, which is the list $(0, 1)$.

A qubit can be in a mix of both (superposition), like

$$|\,\psi\rangle = \alpha\,|\,0\rangle + \beta\,|\,1\rangle,$$



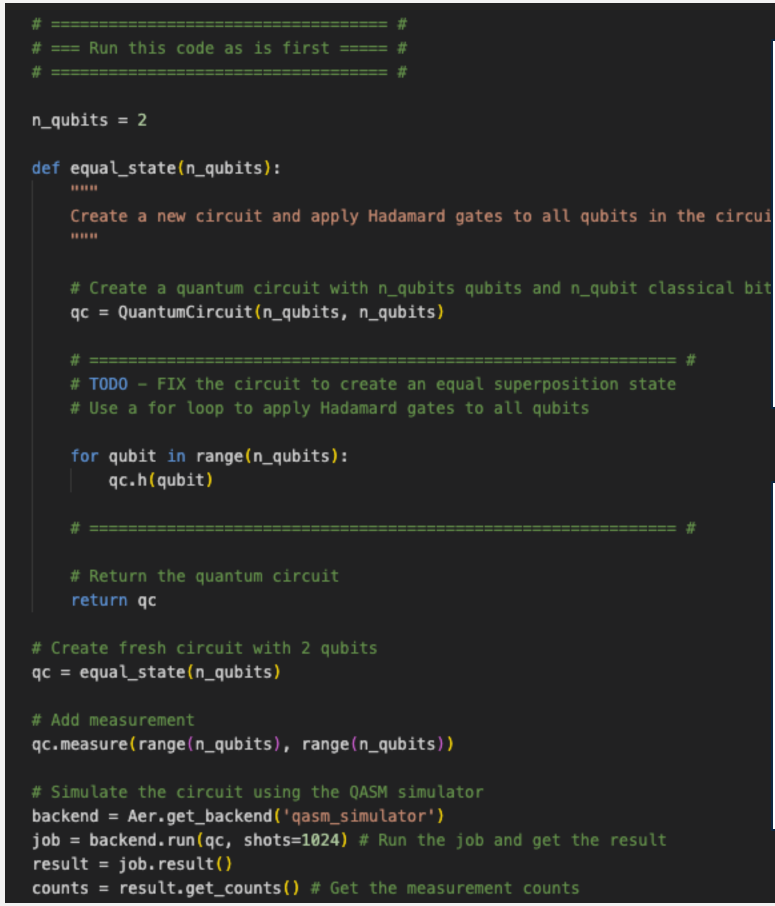


**Food for thought:**

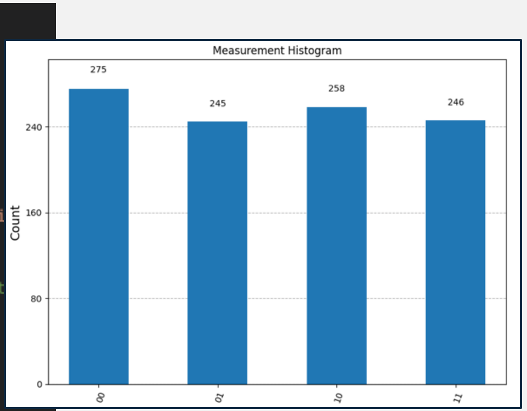
- What happens if you only apply the Hadamard to one qubit?
- Can you predict the probabilities just by looking at the gates?
- Why are the simulated results not perfectly distributed

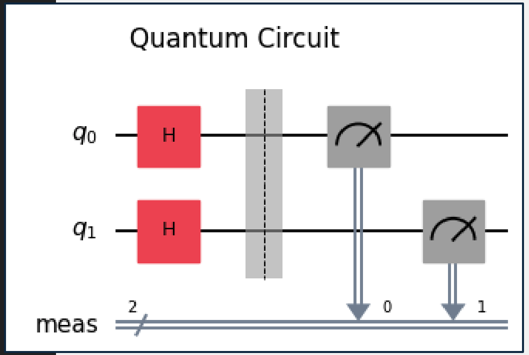
Exercise 1

# Create a Uniform Superposition



```
# ================================== #
# === Run this code as is first ===== #
# ================================== #

n_qubits = 2

def equal_state(n_qubits):
    """
    Create a new circuit and apply Hadamard gates to all qubits in the circui
    """

    # Create a quantum circuit with n_qubits qubits and n_qubit classical bit
    qc = QuantumCircuit(n_qubits, n_qubits)

    # ============================================================ #
    # TODO - FIX the circuit to create an equal superposition state
    # Use a for loop to apply Hadamard gates to all qubits

    for qubit in range(n_qubits):
        qc.h(qubit)

    # ============================================================ #

    # Return the quantum circuit
    return qc

# Create fresh circuit with 2 qubits
qc = equal_state(n_qubits)

# Add measurement
qc.measure(range(n_qubits), range(n_qubits))

# Simulate the circuit using the QASM simulator
backend = Aer.get_backend('qasm_simulator')
job = backend.run(qc, shots=1024) # Run the job and get the result
result = job.result()
counts = result.get_counts() # Get the measurement counts
```





## Objective

- Test students’ ability to apply a Hadamard gate to every qubit to create a uniform superposition.

## Methodology

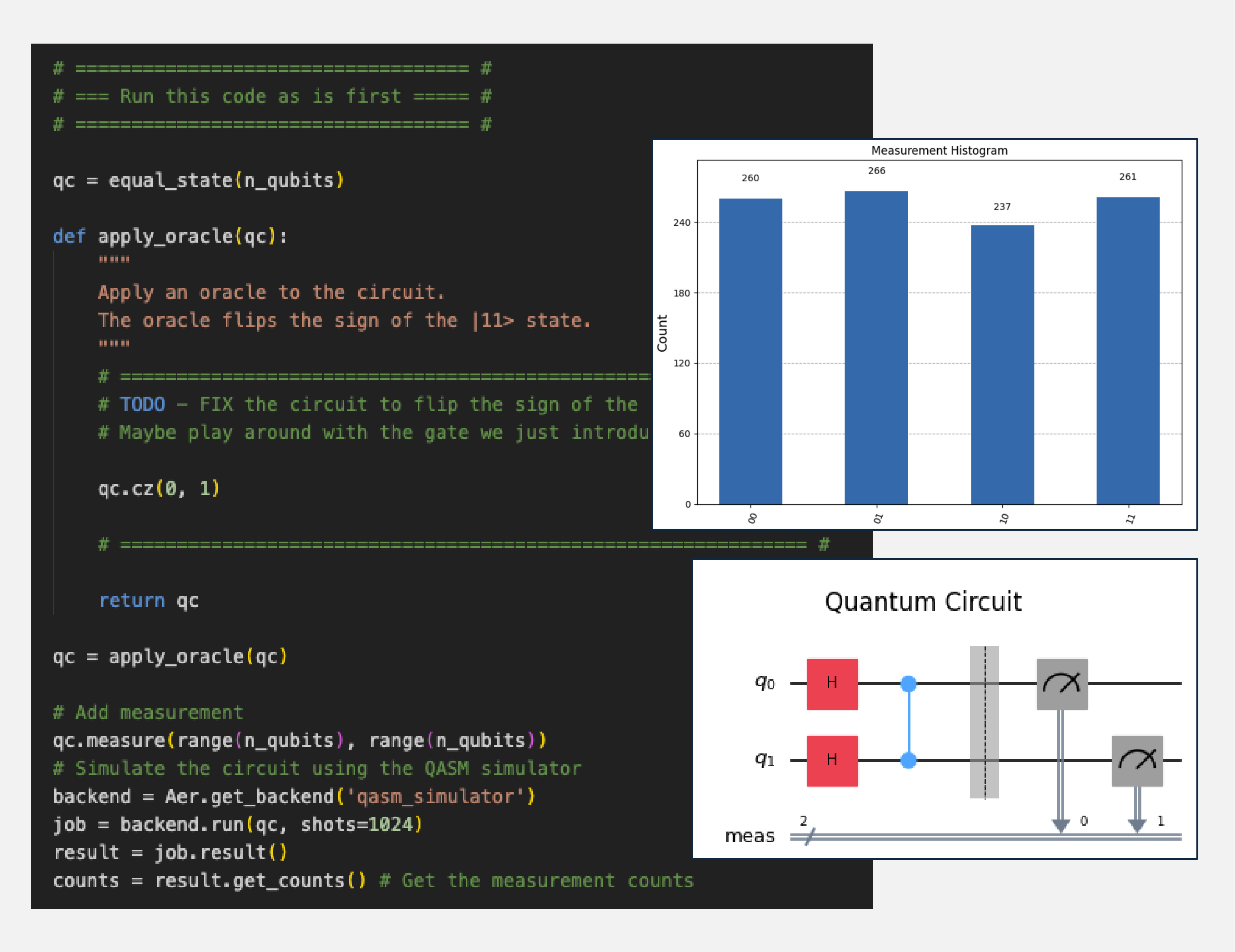
- Use a for-loop over range(n_qubits) and call qc.h(qubit) inside it.

## Aimed Learning Outcomes

- Students write simple loops in Qiskit.
- Students see how multiple H-gates produce an equal superposition.
- Students recognise the role of uniform states in Grover’s algorithm.

Exercise 2

# Oracles as Black Boxes



```
# =================================== #
# === Run this code as is first ===== #
# =================================== #

qc = equal_state(n_qubits)

def apply_oracle(qc):
    """
    Apply an oracle to the circuit.
    The oracle flips the sign of the |11> state.
    """
    # ===============================================
    # TODO - FIX the circuit to flip the sign of the
    # Maybe play around with the gate we just introdu

    qc.cz(0, 1)

    # ============================================================= #

    return qc

qc = apply_oracle(qc)

# Add measurement
qc.measure(range(n_qubits), range(n_qubits))
# Simulate the circuit using the QASM simulator
backend = Aer.get_backend('qasm_simulator')
job = backend.run(qc, shots=1024)
result = job.result()
counts = result.get_counts() # Get the measurement counts
```

## Objective

- Implement a phase-flip oracle that marks a chosen basis state by flipping its sign.

## Methodology

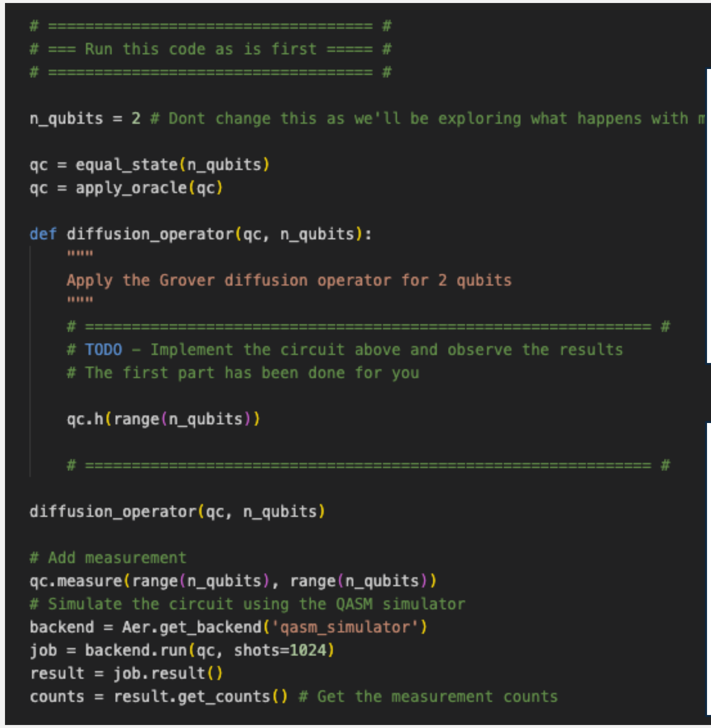
- Insert a controlled-Z gate (qc.cz(0,1)) into the superposed circuit to flip only the |11⟩ amplitude.

## Aimed Learning Outcomes

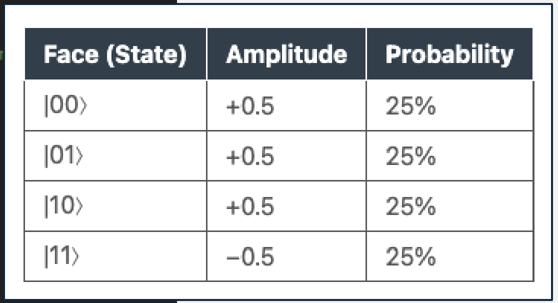
- Students encode the “black-box” oracle as a phase flip.
- Students observe that only the phase (not the probability) changes.
- Students grasp that the oracle is problem-defined, not discovered.

Exercise 3

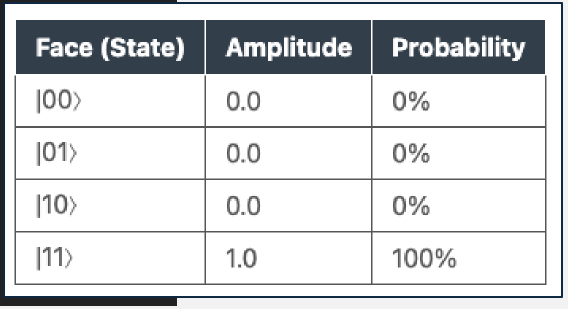
# The Diffusion Operator



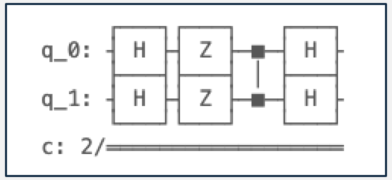
```
# ================================== #
# === Run this code as is first ===== #
# ================================== #

n_qubits = 2 # Dont change this as we'll be exploring what happens with m

qc = equal_state(n_qubits)
qc = apply_oracle(qc)

def diffusion_operator(qc, n_qubits):
    """
    Apply the Grover diffusion operator for 2 qubits
    """
    # ============================================================ #
    # TODO - Implement the circuit above and observe the results
    # The first part has been done for you

    qc.h(range(n_qubits))

    # ============================================================ #

diffusion_operator(qc, n_qubits)

# Add measurement
qc.measure(range(n_qubits), range(n_qubits))
# Simulate the circuit using the QASM simulator
backend = Aer.get_backend('qasm_simulator')
job = backend.run(qc, shots=1024)
result = job.result()
counts = result.get_counts() # Get the measurement counts
```



| Face (State) | Amplitude | Probability |
|---|---|---|
| \|00⟩ | +0.5 | 25% |
| \|01⟩ | +0.5 | 25% |
| \|10⟩ | +0.5 | 25% |
| \|11⟩ | −0.5 | 25% |



| Face (State) | Amplitude | Probability |
|---|---|---|
| \|00⟩ | 0.0 | 0% |
| \|01⟩ | 0.0 | 0% |
| \|10⟩ | 0.0 | 0% |
| \|11⟩ | 1.0 | 100% |



## Objective

- Build and apply Grover’s diffusion operator to amplify the marked state.

## Methodology

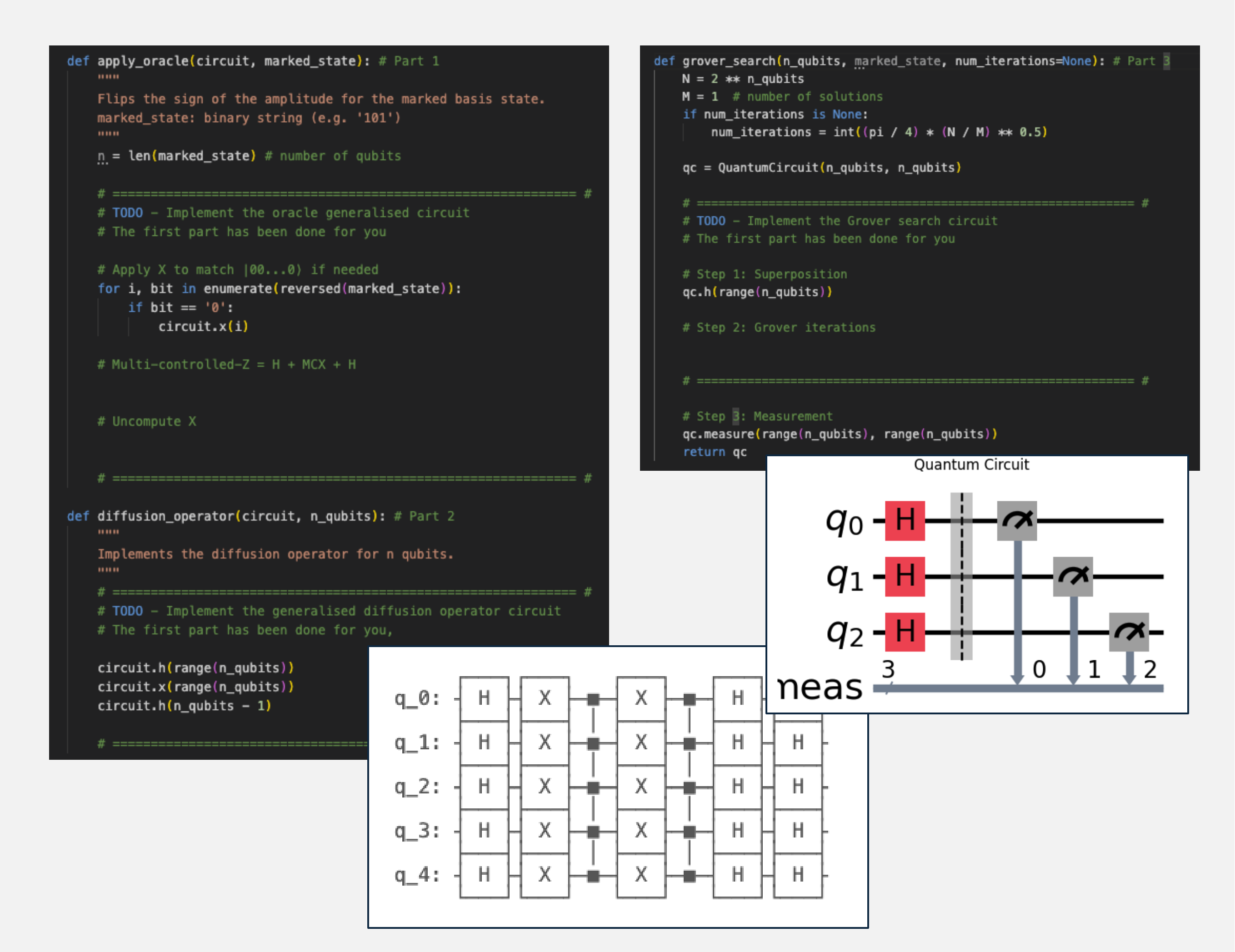
- Apply Hadamards to all qubits.
- Apply Z gates to invert phases about |00…0⟩.
- Use a controlled-Z to flip the global phase.
- Reapply Hadamards to complete the reflection.

## Aimed Learning Outcomes

- Students implement “reflect-about-mean” trick.
- Students see how phase flips become amplitude boosts.
- Students grasp amplitude amplification in one quantum step.

Exercise 4

# The Full Grover Implementation



```python
def apply_oracle(circuit, marked_state): # Part 1
    """
    Flips the sign of the amplitude for the marked basis state.
    marked_state: binary string (e.g. '101')
    """
    n = len(marked_state) # number of qubits

    # ============================================================= #
    # TODO - Implement the oracle generalised circuit
    # The first part has been done for you

    # Apply X to match |00...0⟩ if needed
    for i, bit in enumerate(reversed(marked_state)):
        if bit == '0':
            circuit.x(i)

    # Multi-controlled-Z = H + MCX + H


    # Uncompute X


    # ============================================================= #

def diffusion_operator(circuit, n_qubits): # Part 2
    """
    Implements the diffusion operator for n qubits.
    """
    # ============================================================= #
    # TODO - Implement the generalised diffusion operator circuit
    # The first part has been done for you,

    circuit.h(range(n_qubits))
    circuit.x(range(n_qubits))
    circuit.h(n_qubits - 1)

    # ===================================
```

```python
def grover_search(n_qubits, marked_state, num_iterations=None): # Part 3
    N = 2 ** n_qubits
    M = 1  # number of solutions
    if num_iterations is None:
        num_iterations = int((pi / 4) * (N / M) ** 0.5)

    qc = QuantumCircuit(n_qubits, n_qubits)

    # ============================================================= #
    # TODO - Implement the Grover search circuit
    # The first part has been done for you

    # Step 1: Superposition
    qc.h(range(n_qubits))

    # Step 2: Grover iterations


    # ============================================================= #

    # Step 3: Measurement
    qc.measure(range(n_qubits), range(n_qubits))
    return qc
```

## Objective

- Implement a fully general, multi-qubit Grover search with arbitrary marked states.

## Methodology

- Build oracle and diffusion circuits for n qubits
- Compute optimal iteration count
- Repeat: oracle → diffusion → measure

## Aimed Learning Outcomes

- Generalize key Grover blocks to arbitrary size
- Use formula for optimal Grover iterations
- See full algorithm in action and its speed-up

Conclusion

# Thank You!

**Thank You! We have had a fantastic time building this tutorial so thank you for hosting and we hope that you enjoy it!**

**If you have any further questions don’t hesitate to reach out to us at:**

- **Felix:** felixarkle@icloud.com
- **Seb:** seb.gotto@gmail.com

**Happy quantum computing,**
**Felix and Seb :)**