# Introduction to Quantum Computing

> "I think I can safely say that nobody understands quantum mechanics."

Richard Feynman

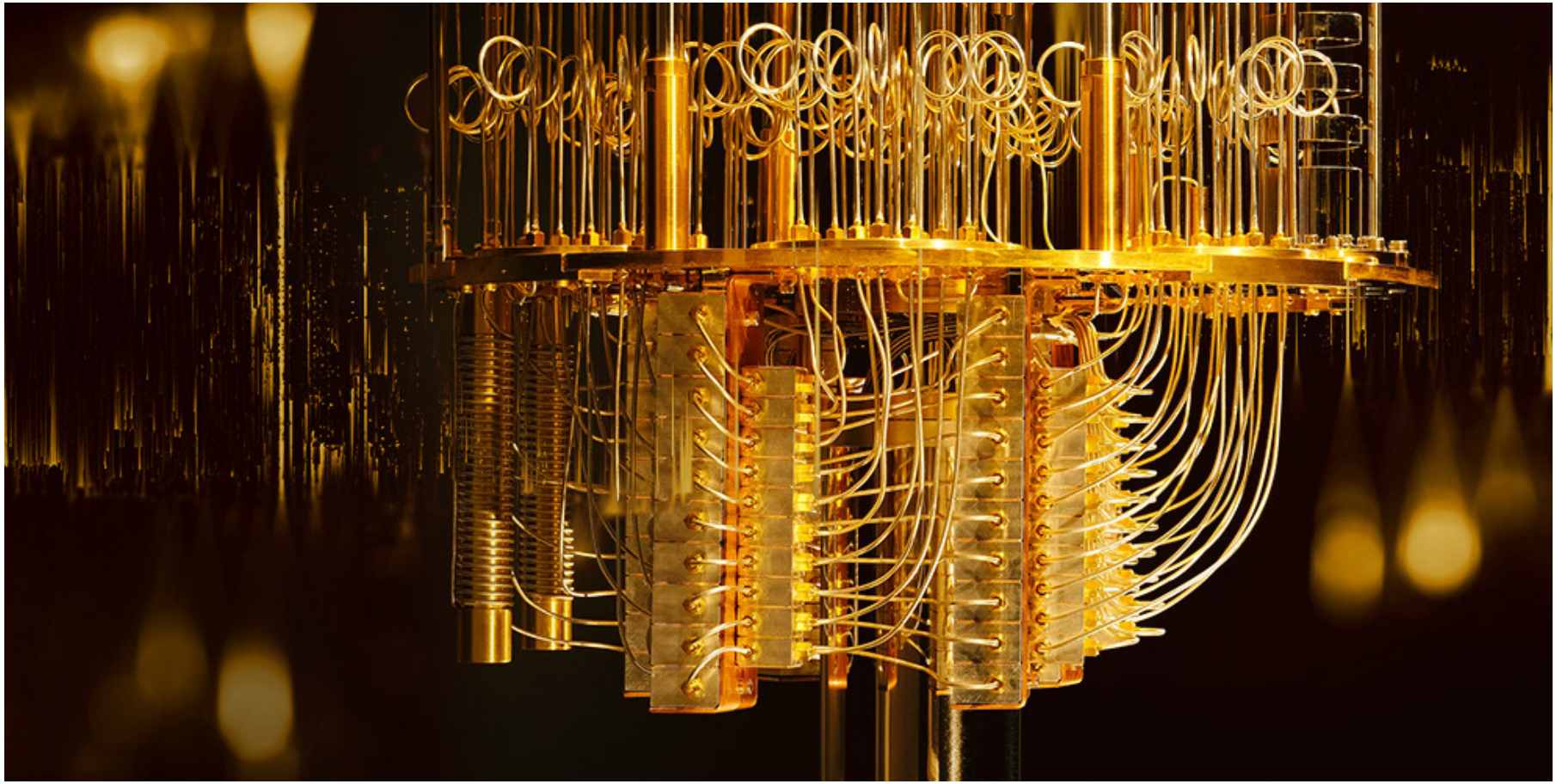
> "If you are not completely confused by quantum mechanics, you do not understand it."

John Wheeler

> "I do not like it, and I am sorry I ever had anything to do with it."

Erwin Schrödinger

- Potentially super polynomial speedup (Shor)
- Simulation of qunatum mechanical scenarios



## Dirac Notation/Braket Notation

- Kets: $|\phi\rangle = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \cdots \\ \phi_n \end{pmatrix}$

- Inner Product: $< \phi|\psi >= (\ \phi_1 \qquad \phi_2 \qquad \dots \qquad \phi_n\ ) \begin{pmatrix} \psi_1 \\ \psi_2 \\ \dots \\ \psi_n \end{pmatrix} = \phi_1\psi_1 + \phi_2\psi_2 + \dots + \phi_n\psi_n$

- Tensor product:

$$\begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} \otimes \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix} = \begin{pmatrix} \phi_1\psi_1 \\ \phi_1\psi_2 \\ \phi_2\psi_1 \\ \phi_2\psi_1 \end{pmatrix}$$

- We write:

$$|\phi\rangle \otimes |\psi\rangle = |\phi\rangle\,|\psi\rangle = |\phi\psi\rangle$$

- Some special kets:
  - $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
  - $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
  - $|+\rangle = \begin{pmatrix} h \\ h \end{pmatrix}$
  - $|-\rangle = \begin{pmatrix} h \\ -h \end{pmatrix}$
  - where $h = \frac{1}{\sqrt{2}}$

- Examples:

- $|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$
- $|11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$
- $|101\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$

## The Qubit

- Smallest unit of quantum information
- Superpostion of $|0\rangle$ and $|1\rangle$
  - Point on the Bloch sphere
- Written as: $|\phi\rangle = \alpha |0\rangle + \beta |1\rangle$ (where $\alpha, \beta \in \mathbb{C}$)
  - so that: $\alpha^2 + \beta^2 = 1$
  - Alternative representation: $|\phi\rangle = r_1 e^{i\theta_1} |0\rangle + r_2 e^{i\theta_2} |1\rangle$

In [1]:
```
from IPython.display import display
import ipywidgets as widgets
from ipywidgets import Layout
from qiskit.visualization import plot_bloch_vector
import cmath
%matplotlib inline
```

```
def bloch_sphere_demo():
    a = widgets.FloatSlider(description='theta', min=0, max=2*math.pi, default=0.0)
    b = widgets.FloatSlider(description='phi', min =0, max=2*math.pi, default=0.0)

    def f(theta, phi):
        x = math.sin(theta)*math.cos(phi)
        y = math.sin(theta)*math.sin(phi)
        z = math.cos(theta)
        # Remove horrible almost-zero results
        if abs(x) < 0.0001:
            x = 0
        if abs(y) < 0.0001:
            y = 0
        if abs(z) < 0.0001:
            z = 0
        display(plot_bloch_vector([x,y,z]))


    def g(theta, phi):
        print("alpha = {}\nbeta = {}".format(math.cos(theta/2.0), math.e**(complex(0, 1) * phi)*math.sin(theta/2.0)))


    out = widgets.interactive_output(f, {'theta': a, 'phi': b})
    out1 = widgets.interactive_output(g, {'theta': a, 'phi': b})

    return widgets.HBox([widgets.VBox([a, b, out1], layout=Layout(width='50%')), out])
```

In [7]:
```
bloch_sphere_demo()
```

```
HBox(children=(VBox(children=(FloatSlider(value=0.0, description='theta', max=6.283185307179586), FloatSlider(…
```

- Representation is limited as not all states are independent qubits
- Entanglement = non-seperable qubits
  - Example Bell-state: $|\Phi^+\rangle = h(|00\rangle + |11\rangle) = \begin{pmatrix} h \\ 0 \\ 0 \\ h \end{pmatrix}$

# State changes: gates

- Programming as circuit: arrangement of gates
- Gate is unitary transformation on fixed number of qubits (#inputs = #outputs)
- Gates are always reversible (wlog)
  - Mathematically expressed as matrix multiplication
  - Most notable gates are:
    - Hadamard-gate: $H = \begin{pmatrix} h & h \\ h & -h \end{pmatrix}$
    - Not-Gate: $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
    - CNot-Gate: $CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
- Matrix $M$ is unitary iff: $M^{-1} = M^\dagger \Leftrightarrow M^\dagger M = MM^\dagger = I$
- every 1 qubit manipulation can be thought of as rotation on the Bloch sphere
- CX-gate used to entangle: $CX\left|+\right\rangle\left|0\right\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} h \\ 0 \\ h \\ 0 \end{pmatrix} = \begin{pmatrix} h \\ 0 \\ 0 \\ h \end{pmatrix} = \left|\Phi^+\right\rangle$
- Universal-set of gates: every operation can be simulated with arbitrarily small error using only gates from this set

# Measurements

- In order to "know" the state of a qubit it has to be measured
- Measurements have 2 effects:
  - return a classical bit as result
  - collapse the qubit to basis state
- For qubit $\left|\phi\right\rangle = \alpha\left|0\right\rangle + \beta\left|1\right\rangle$

- measurement will return 0 with probability $|\alpha|^2$ (1 with prob. $|\beta|^2$)
- state will collapse to measured state (either $|0\rangle$ or $|1\rangle$)

- Example: $|\Phi^+\rangle = \begin{pmatrix} h \\ 0 \\ 0 \\ h \end{pmatrix}$
  - will return 0 or 1 with probability 0.5 each
  - resulting state will be either either $|00\rangle$ or $|11\rangle$
  - notice that the result of the measurement determines the state of the second qubit

# Example: Deutsch Algorithm

## Classic Version

Demonstrating the purpose of the algorithm in standard python.

In [2]:
```
def isBalanced(f):
    return f(False) != f(True)
```

In [9]:
```
isBalanced(lambda x: not x)
```

Out[9]:
```
True
```

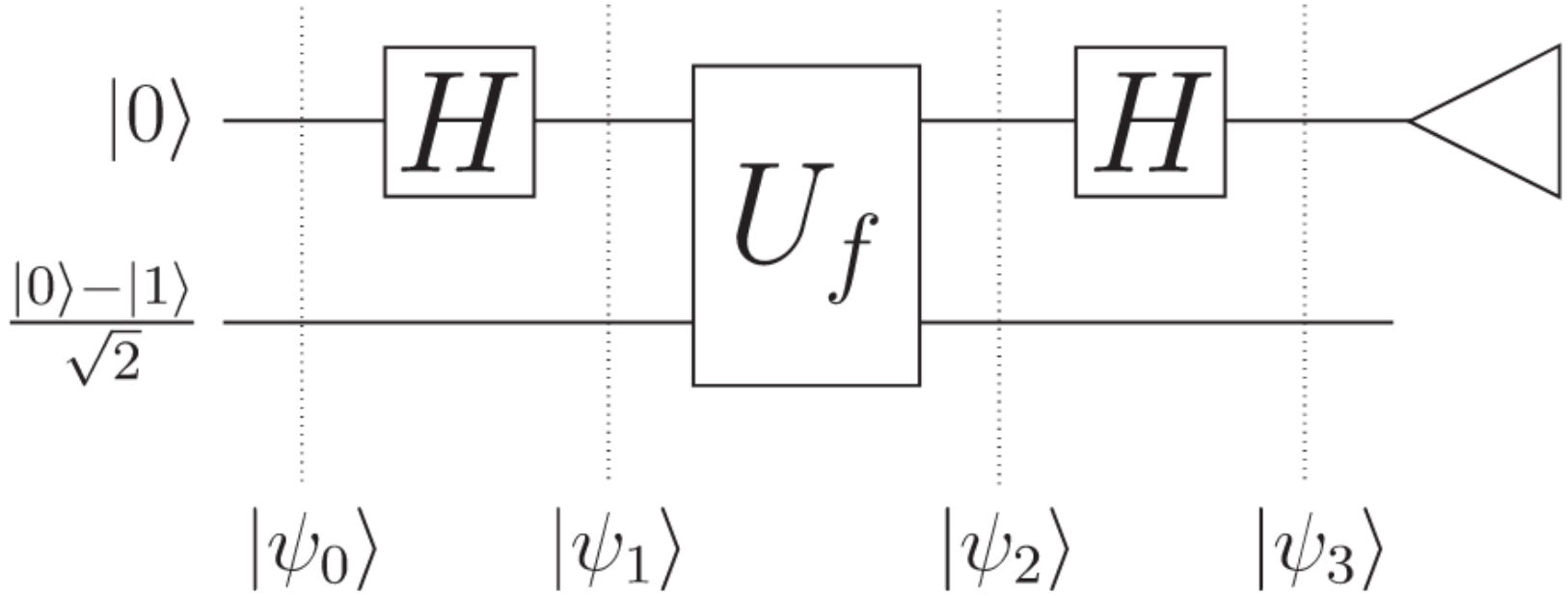
In [10]:
```
isBalanced(lambda x: False)
```

Out[10]:
```
False
```

### Deutsch Problem

Given a function $f : \{0, 1\} \to \{0, 1\}$ and the guarantee that this function is either constant or balanced, determine whether it is balanced or not.

# Deutsch-Josza explanation



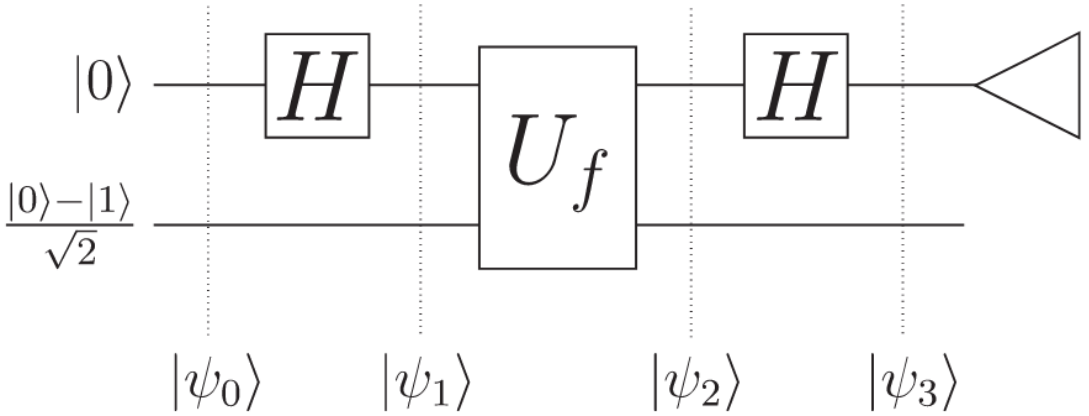

- f is arbitrary function $f : \{0, 1\} \to \{0, 1\}$
- f may not be reversible thus not applicable in circuit
- workaround: create unitary mapping $U_f : |x\rangle\,|y\rangle \to |x\rangle\,|f(x) \oplus y\rangle$

$$U_f : |x\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \rightarrow |x\rangle \left( \frac{f(x) \oplus |0\rangle - f(x) \oplus |1\rangle}{\sqrt{2}} \right)$$

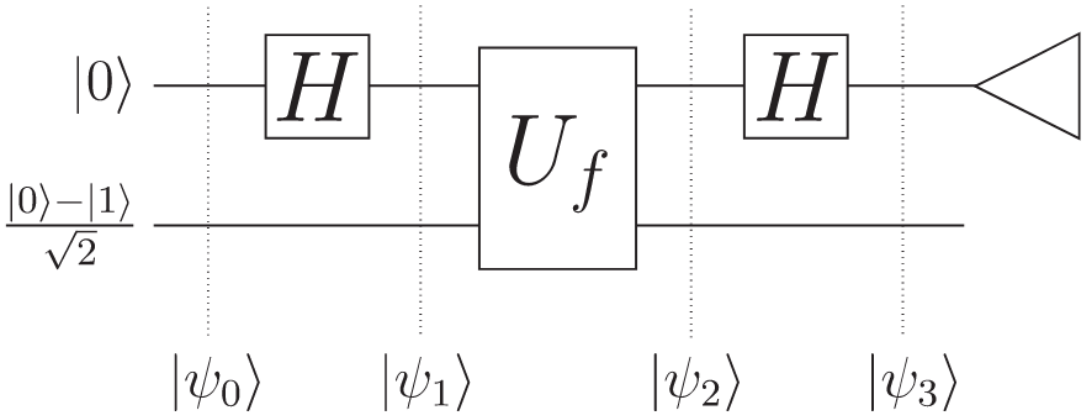
$$f(x) = 0 : \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$f(x) = 1 : \frac{|1\rangle - |0\rangle}{\sqrt{2}} = - \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

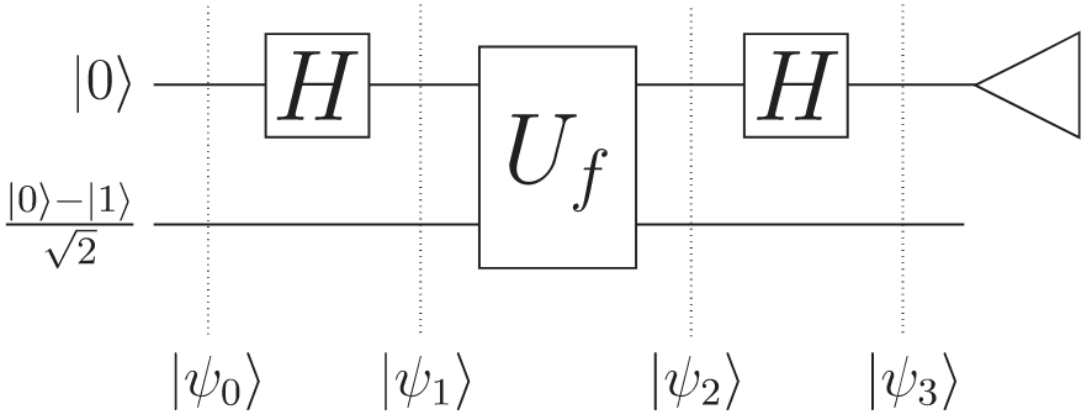
$$U_f : |x\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \rightarrow |x\rangle \left( \frac{f(x) \oplus |0\rangle - f(x) \oplus |1\rangle}{\sqrt{2}} \right) = (-1)^{f(x)} |x\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$



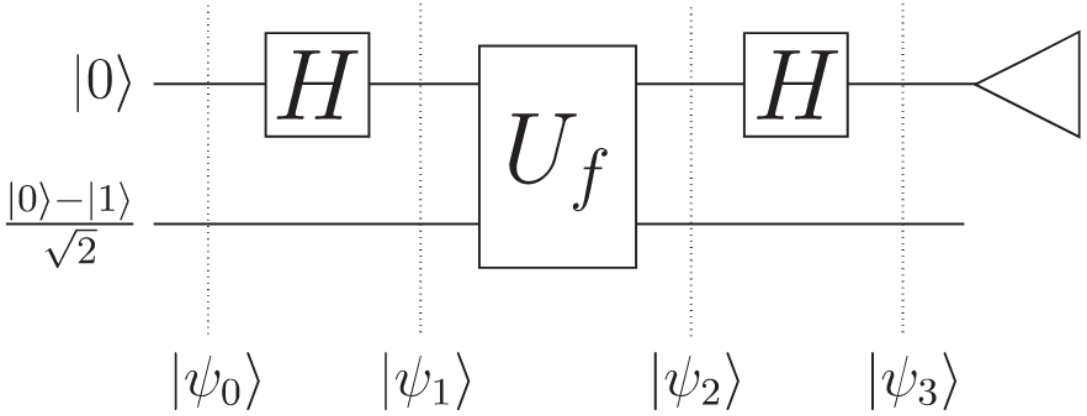

$$|\phi_0\rangle = |0\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$



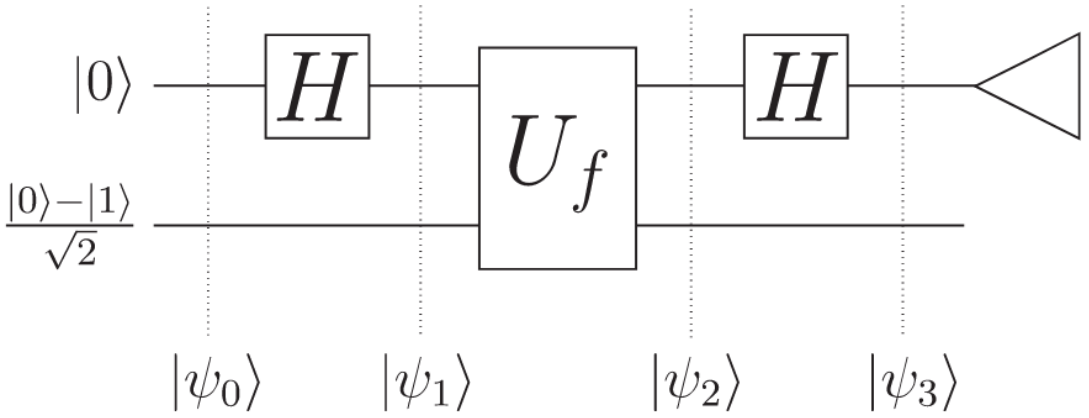

$$|\phi_1\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) =$$
$$\frac{1}{\sqrt{2}}|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{1}{\sqrt{2}}|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$



$$|\phi_2\rangle = \frac{(-1)^{f(0)}}{\sqrt{2}}|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{(-1)^{f(1)}}{\sqrt{2}}|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) =$$
$$\left(\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) =$$
$$(-1)^{f(0)}\left(\frac{|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

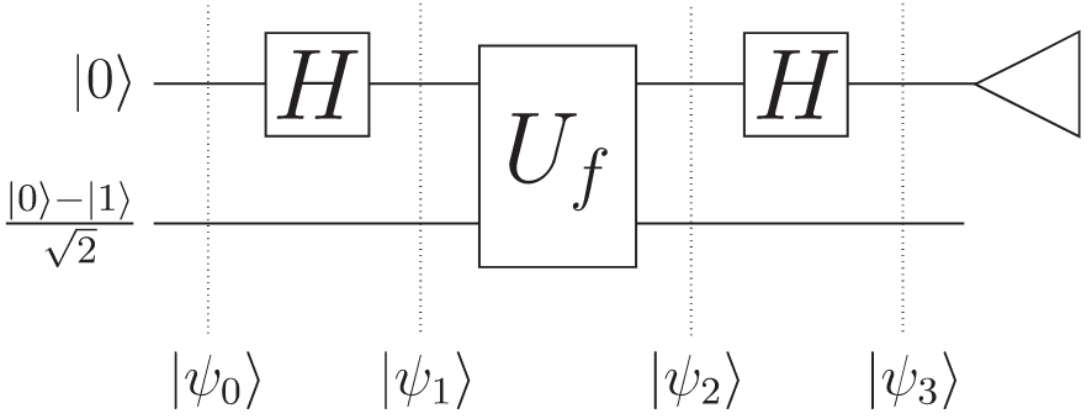



$$f(0)\oplus f(1) = 0 : (-1)^{f(0)}\left(\frac{|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) =$$
$$(-1)^{f(0)}\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$
$$f(0)\oplus f(1) = 1 : (-1)^{f(0)}\left(\frac{|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) =$$
$$(-1)^{f(0)}\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$




$$f(0) \oplus f(1) = 0 : |phi_3\rangle = (-1)^{f(0)} |0\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

$$f(0) \oplus f(1) = 1 : |phi_3\rangle = (-1)^{f(0)} |1\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$



- measure 0: f is constant
- measure 1: f is balanced

## Qiskit Version

Now showing the quantum version of the algorithm as implemented in qiskit.

Import some libraries and choose a backend for qiskit. In this case we use a simulator and do not run this on a real qComputer.

In [3]:
```
import numpy as np
from qiskit import(
  QuantumCircuit,
  execute,
  Aer)
from qiskit.visualization import plot_histogram, plot_state_city
from qiskit.quantum_info.operators import Operator
from qiskit.providers.fake_provider import FakeBoeblingen
import math
import matplotlib.pyplot as plt
%matplotlib inline

def xor(value):
  return value[0] != value[1]

def constFalse(value):
  return False

def constTrue(value):
  return True

def intHalf(value):
  if not ('0b' in value):
      value = '0b' + value
  return int(value, 2) < math.pow(2, len(value) - 3)

def isEven(value):
  return value[-1] == '0'

def runCircuit(circuit, add_measurements=True):
    # Use Aer's qasm_simulator
```

```
    simulator = Aer.get_backend('qasm_simulator')

    circuitcopy = circuit.copy()
    if add_measurements:
        circuitcopy.measure(range(circuit.num_qubits), range(circuit.num_qubits))

    # Execute the circuit on the simulator with error characteristics of the boeblingen backend
    job = execute(circuitcopy, simulator, shots=10024)

    # Grab results from the job
    result = job.result()

    # Returns counts
    counts = result.get_counts(circuitcopy)
    return counts

def showAmplitudes(circuit):
    # Use Aer's qasm_simulator
    simulator = Aer.get_backend('statevector_simulator')

    # Grab results from the job
    result = simulator.run(circuit).result()

    # Returns counts
    statevector = result.get_statevector()
    res = [i.real * abs(i.real) for i in statevector.data]

    bits = [("0" * circuit.num_qubits + bin(i)[2:])[-circuit.num_qubits:] for i in range(2**circuit.num_qubits)]
    plt.bar(bits,res, color='midnightblue')
    plt.show()

def toBool(value):
    return value[0] == '0'
```

We use a helper function to generate a unitary operator from the given function. This performs the operation: $|xy\rangle \rightarrow |x\rangle\,|y \oplus f(x)\rangle$

In [4]:

```
def getOracle(f):
  m = []
  mSize = int(math.pow(2, n+1))
  for i in range(int(math.pow(2, n))):
    idx = i * 2
```

```
    bitString = bin(idx)
    bitString = bitString[2:]
    bitString = "0" * (n + 1 - len(bitString)) + bitString
    flip = f(bitString[:-1]) #"{idx:0$lenBitstring}"
    line = np.zeros(mSize)
    nextLine = np.zeros(mSize)
    if flip:
      line[idx] = 1
      nextLine[idx + 1] = 1
    else:
      nextLine[idx] = 1
      line[idx + 1] = 1
    m.append(line)
    m.append(nextLine)
  return Operator(m)
```

We define the arity and the function we want to examine.

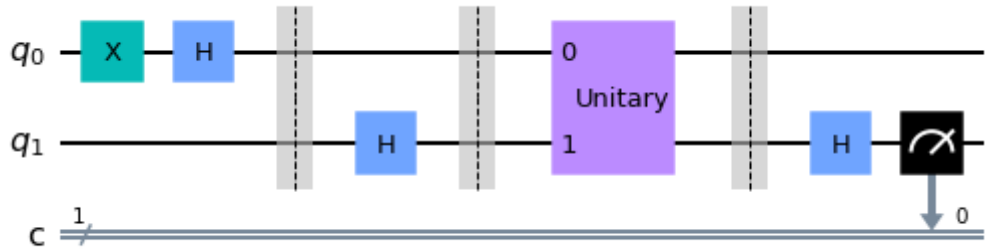
In [5]:
```
#define the arity of the function
n = 1

def f(value):
    #your function
    #return toBool(value)
    #return value[3] == '0'
    return False
```

Create the circuit according to the Deutsch Algorithm.

In [14]:
```
oracle = getOracle(f)

circuit = QuantumCircuit(n+1, n)

circuit.x(0)
circuit.h(0)
circuit.barrier()

for i in range(n):
  circuit.h(i + 1)

circuit.barrier()
```

```
circuit.append(oracle, range(n+1))
circuit.barrier()

for i in range(n):
  circuit.h(i + 1)

# measure all qubits except the first one
for i in range(n):
  circuit.measure([i + 1], [i])

# Draw the circuit
circuit.draw(output='mpl')
```

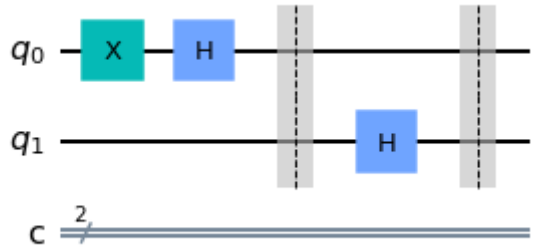
Out[14]:




In [15]:

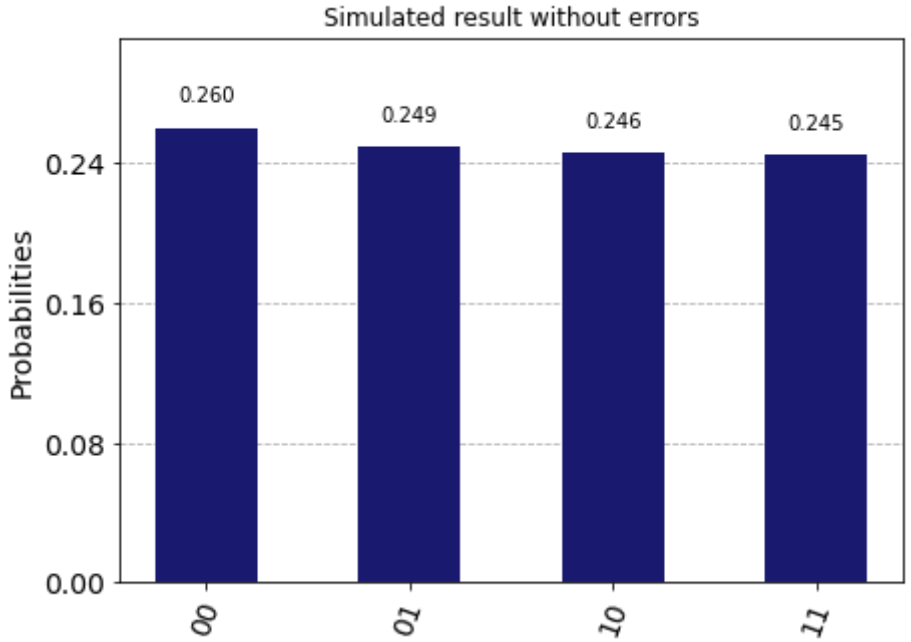
```
circuit = QuantumCircuit(n+1, n+1)

circuit.x(0)
circuit.h(0)
circuit.barrier()

for i in range(n):
  circuit.h(i + 1)

circuit.barrier()
circuit.draw(output='mpl')
```

Out[15]:

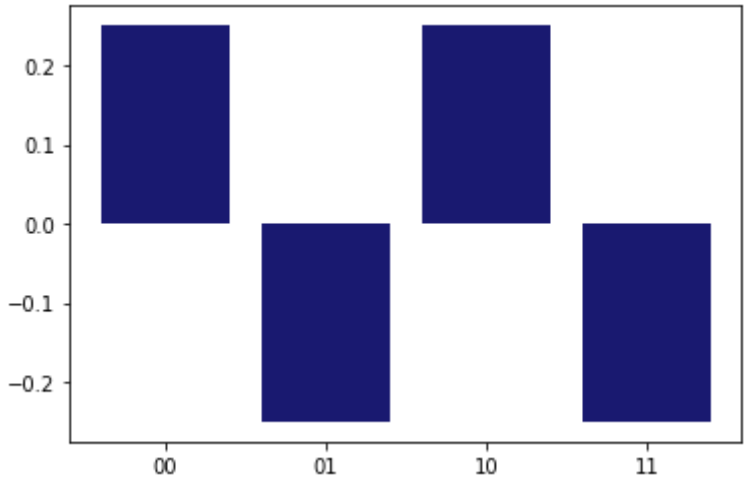



In [16]:
```
plot_histogram(runCircuit(circuit), title='Simulated result without errors', color='midnightblue')
```

Out[16]:

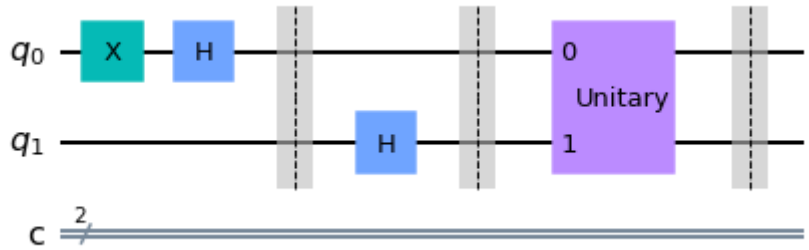



In [17]:
```
showAmplitudes(circuit)
```



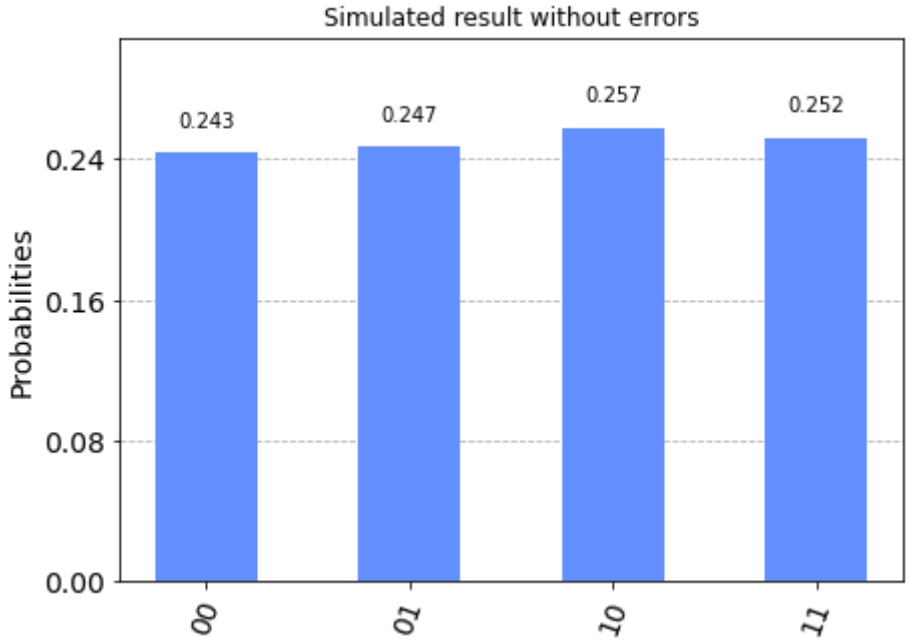

In [18]:
```
oracle = getOracle(f)

circuit.append(oracle, range(n+1))
circuit.barrier()
circuit.draw(output='mpl')
```

Out[18]:

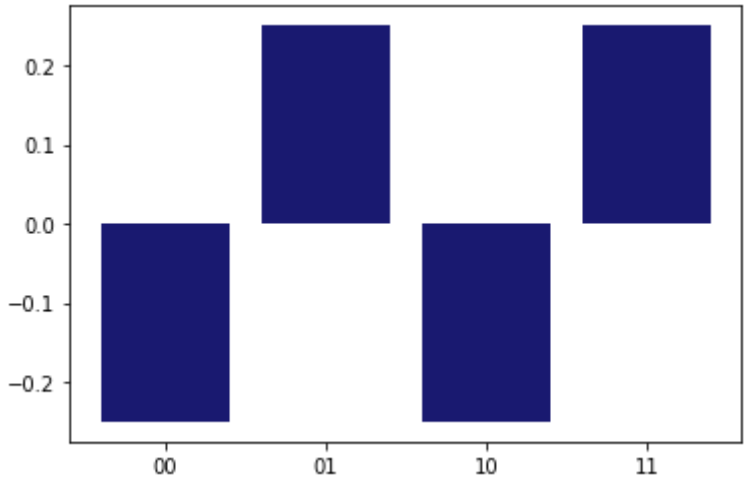



In [19]:
```
plot_histogram(runCircuit(circuit), title='Simulated result without errors')
```

Out[19]:

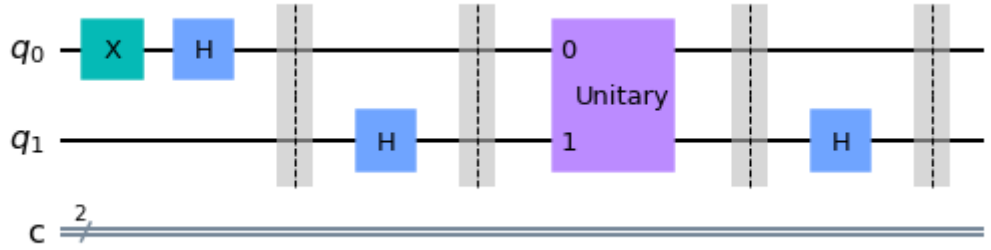



In [20]:
```
showAmplitudes(circuit)
```



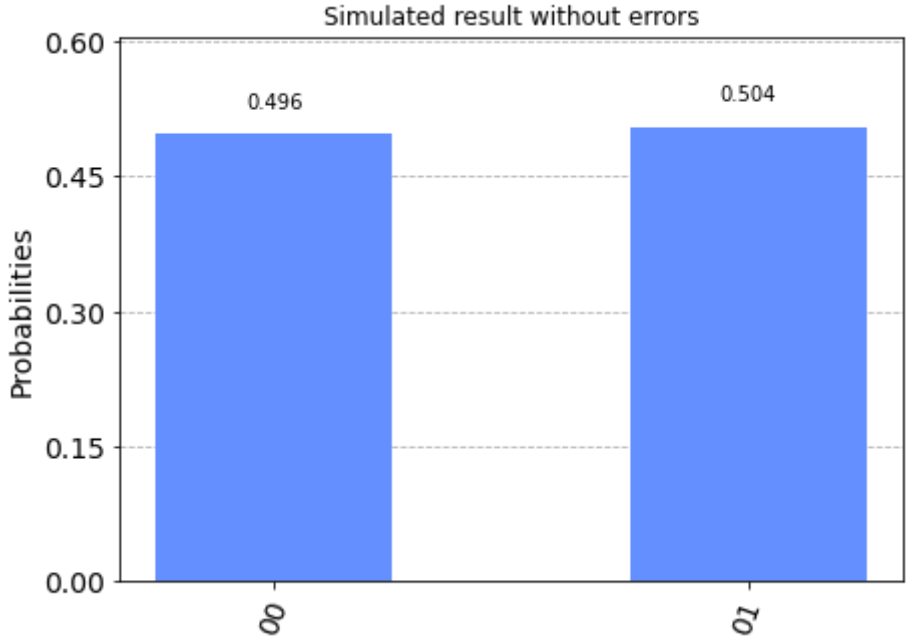

In [21]:
```
for i in range(n):
  circuit.h(i + 1)
circuit.barrier()
circuit.draw(output='mpl')
```

Out[21]:




In [22]:
```
plot_histogram(runCircuit(circuit), title='Simulated result without errors')
```

Out[22]:




In [24]:
```
#define the arity of the function
n = 1

def f(value):
    #your function
    #return not toBool(value)
    #return value[2] == '0'
    return False
```

```
#xor, constFalse, constTrue, intHalf, isEven

oracle = getOracle(intHalf)

circuit = QuantumCircuit(n+1, n)

circuit.x(0)
circuit.h(0)
circuit.barrier()

for i in range(n):
  circuit.h(i + 1)

circuit.barrier()

circuit.append(oracle, range(n+1))
circuit.barrier()

for i in range(n):
  circuit.h(i + 1)

# measure all qubits except the first one
for i in range(n):
  circuit.measure([i + 1], [i])

# Use Aer's qasm_simulator
simulator = Aer.get_backend('qasm_simulator')

# Execute the circuit on the qasm simulator
job = execute(circuit, simulator, shots=1)

# Grab results from the job
result = job.result()

# Returns counts
counts = result.get_counts(circuit)
plot_histogram(counts, title='Simulated result without errors')
print("Measurements:",counts)
print("Is Balanced:")
('0' * n) not in counts
```

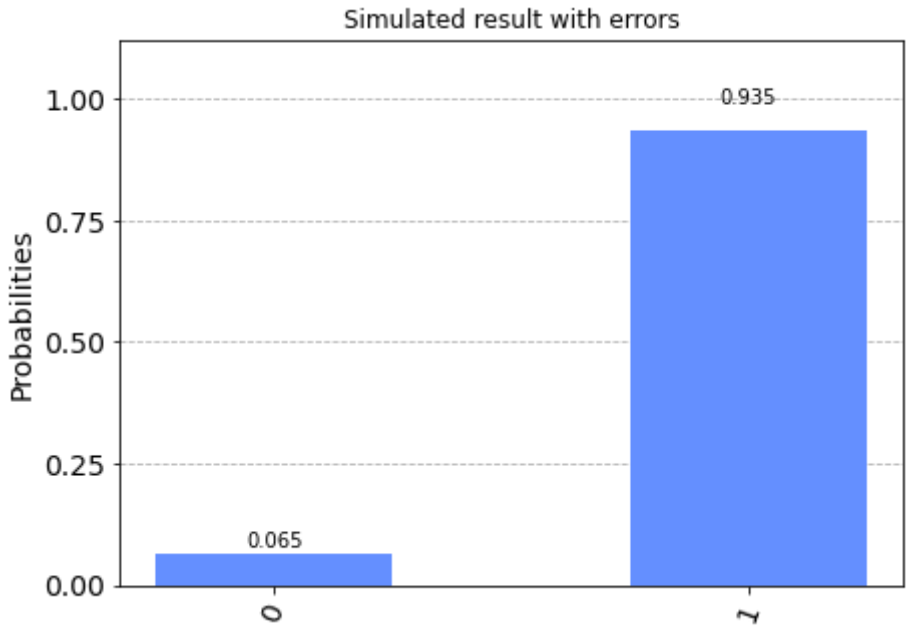
```
Measurements: {'1': 1}
Is Balanced:
```

Out[24]: True

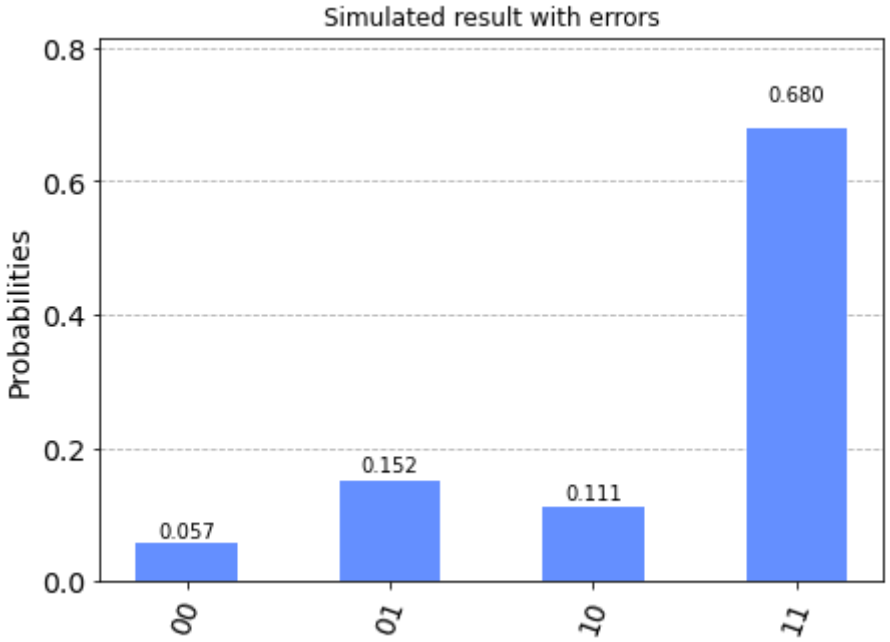
## Simulation with NISQ-errors

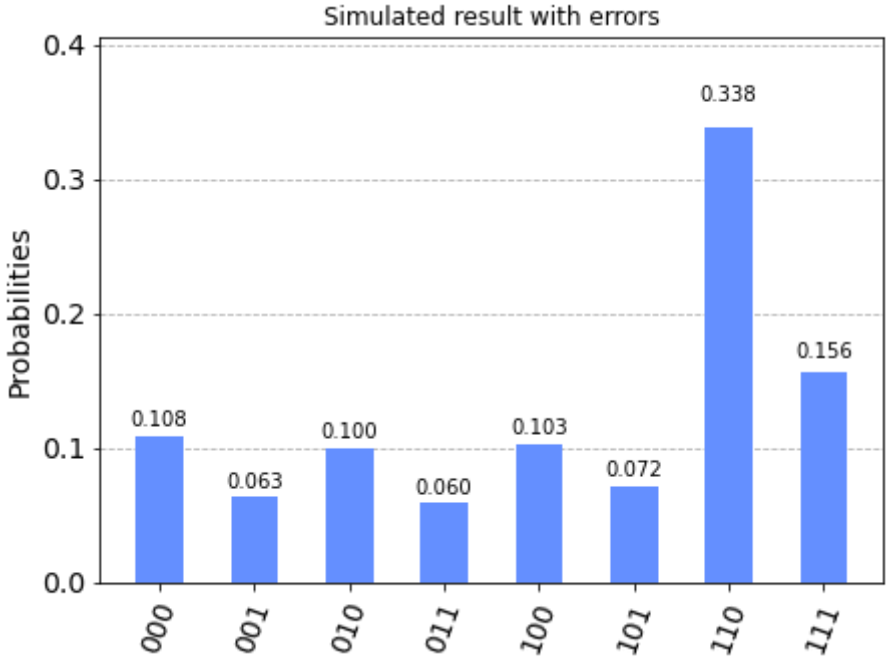
In [25]:
```
device_backend = FakeBoeblingen()
b_simulator = simulator.from_backend(device_backend)

# Execute the circuit on the simulator with error characteristics of the boeblingen backend
job = execute(circuit, b_simulator, shots=1024)

# Grab results from the job
result = job.result()

# Returns counts
counts = result.get_counts(circuit)
plot_histogram(counts, title='Simulated result with errors')
```

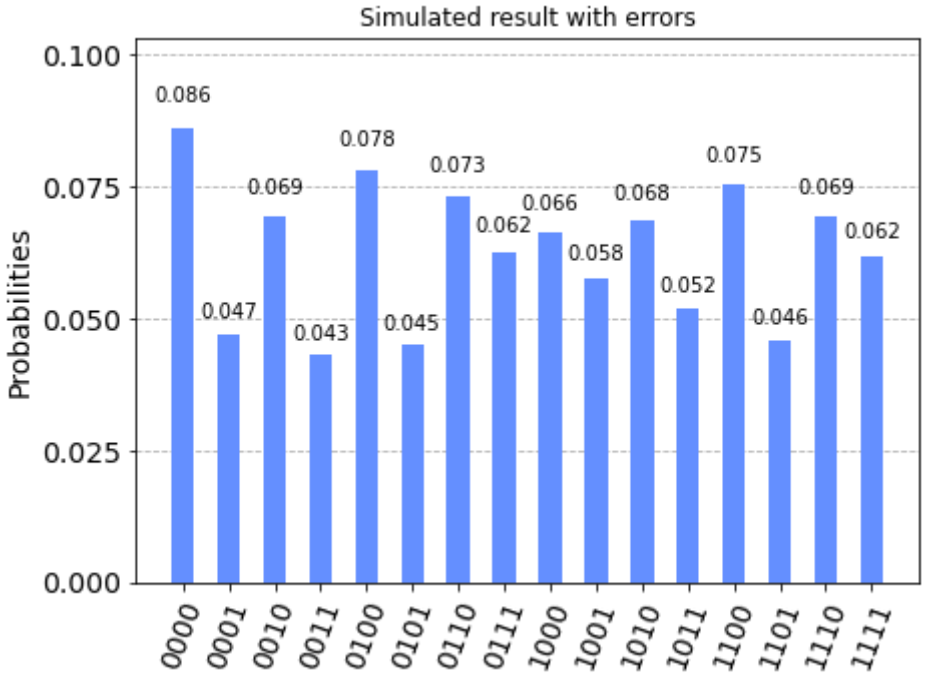
Out[25]:

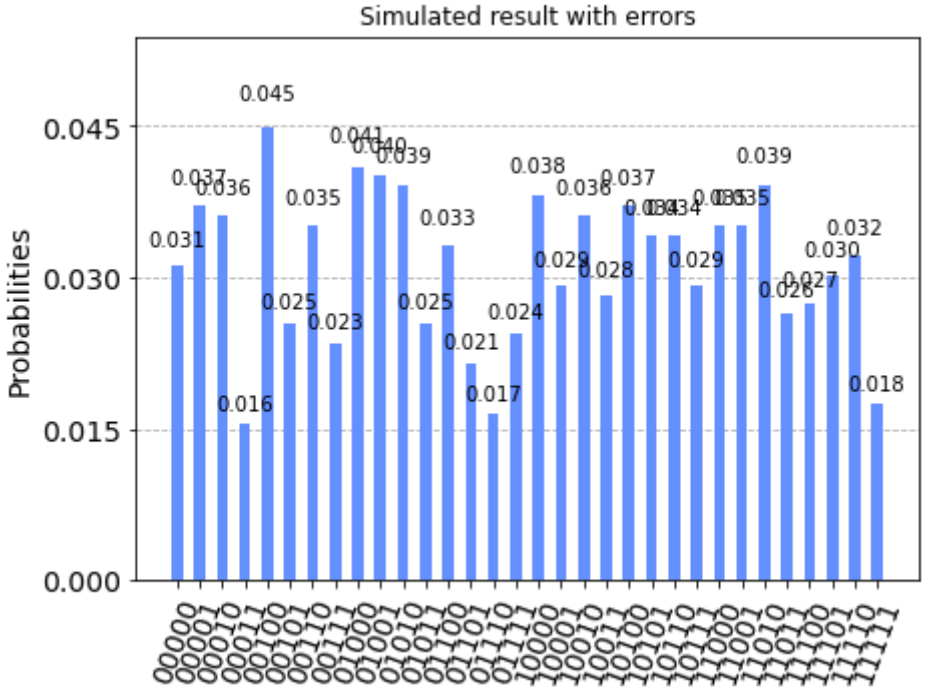



## Demonstrating effect of error rates


Simulated result with errors
Probabilities
0.0
0.2
0.4
0.6
0.8
0.057
0.152
0.111
0.680
00
01
10
11


Simulated result with errors
Probabilities
0.0
0.1
0.2
0.3
0.4
0.108
0.063
0.100
0.060
0.103
0.072
0.338
0.156
000
001
010
011
100
101
110
111


Simulated result with errors
Probabilities
0.100
0.075
0.050
0.025
0.000
0.086
0.047
0.069
0.043
0.078
0.045
0.073
0.062
0.066
0.058
0.068
0.052
0.075
0.046
0.069
0.062
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111


Simulated result with errors
Probabilities
0.045
0.030
0.015
0.000