Assignment 06: Quantum algorithms

Objectives

- You will execute a prime factorization (Shor's algorithm) and you will learn how to browse a database with quantum power (Grover's algorithm)
- You will accomplish a quantum teleportation

Requirements

- Notebook or Desktop Computer with access to the internet
- Access to IBM Quantum Lab (https://lab.quantum-computing.ibm.com/)

Solution Steps

Classifying computer algorithms
In computer science an algorithm describes a finite sequence of well-defined logical instructions for solving mathematical problems or for generating computation.

For example, an algorithm that decides if a number is even only needs to look at the last digit in that number. In this case, the "input" is a number, and the output is "Even" or "Odd" [8].

Input	**Operation**	**Output**

`13`	`let x be the rightmost digit`
	`if x is 0, 2, 4, 6 or 8:`
	`        output Even`
	`else:`
	`        output Odd`
		`Odd`

In general we can note that Quantum Computers usually are able to solve mathematical problems faster than Classical Computers and that a Quantum Computer needs quantum algorithms, such as Grover's search algorithm, for carrying out internal mathematical operations.

Quantum algorithms
There are several algorithms already developed for quantum computing including Grover's for searching an unstructured database and Shor's for factoring large numbers.

1. Shor's algorithm
In 1994, Peter Shor formulated his famous quantum algorithm that factors integers exponentially faster than any known classical algorithm. It caused a lot of excitement beyond academia because modern encryption builds upon the fact that it takes thousands of years to find the prime factors of large numbers. If we could factor such numbers in a few minutes, prime-factor-based encryption collapsed like a house of cards [1].
To understand which impact Shor's algorithm could have on our daily life's please watch the following video: https://youtu.be/zw1KHLOOlA8

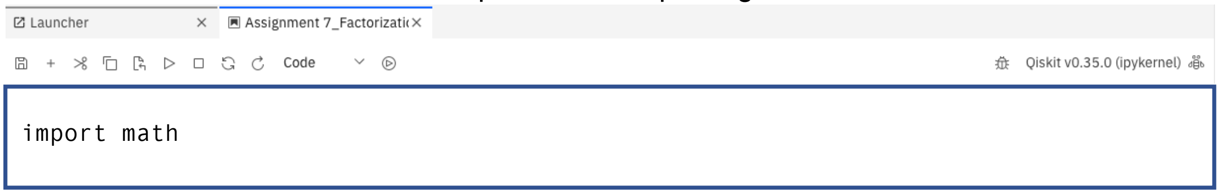
Do you want a taste of prime factorization?
Try to solve the following factorization problems without a computer.

  Find the prime factors of the number 21
  Find the prime factors of the number 221
  Find the prime factors of the number 2231
  Find the prime factors of the number 22523

I am pretty sure you'll solve the first task very easily. Maybe, you come up with the solution right away. When you cope with the second problem, try to count the number of attempts. You'll likely need a lot more attempts to find the factors for 2231. And, I believe you won't be able to find those of 22523 at all.

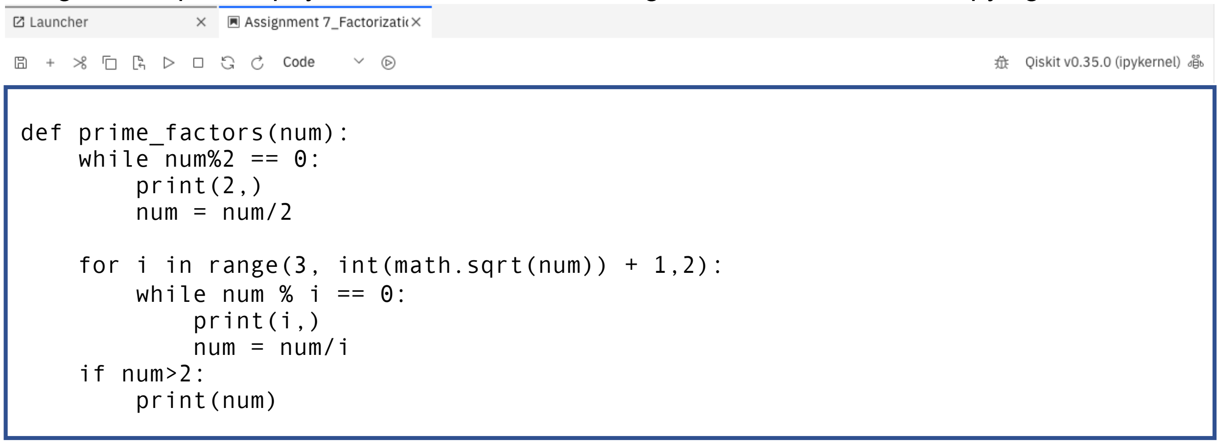
For small numbers like in the example above we can use Classical Computers as well as Quantum Computers to solve the problem.

Let's start with defining a function that enables factorization on a Classical Computer. Open a new file in IBM Quantum Lab and import the math package.



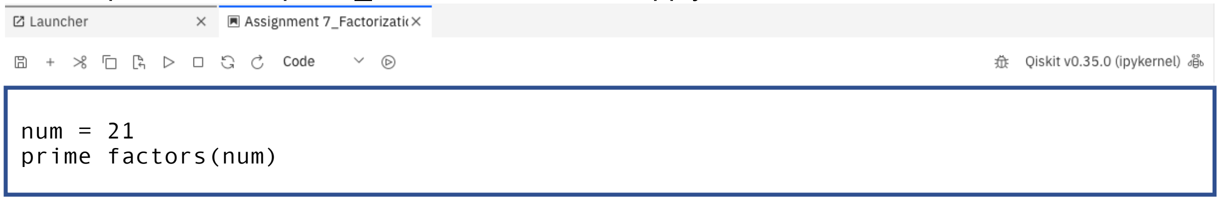
```
import math
```

Define the function "prime_factors" which identifies the prime factors. As in the previous assignments please pay attention to the formatting of the function when copying it.



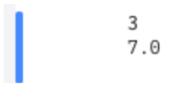
```
def prime_factors(num):
    while num%2 == 0:
        print(2,)
        num = num/2

    for i in range(3, int(math.sqrt(num)) + 1,2):
        while num % i == 0:
            print(i,)
            num = num/i
    if num>2:
        print(num)
```

We can now define the number we want to factorize and assign it to the variable "num". In the next step we use the prime_factors function and apply it on the variable "num".



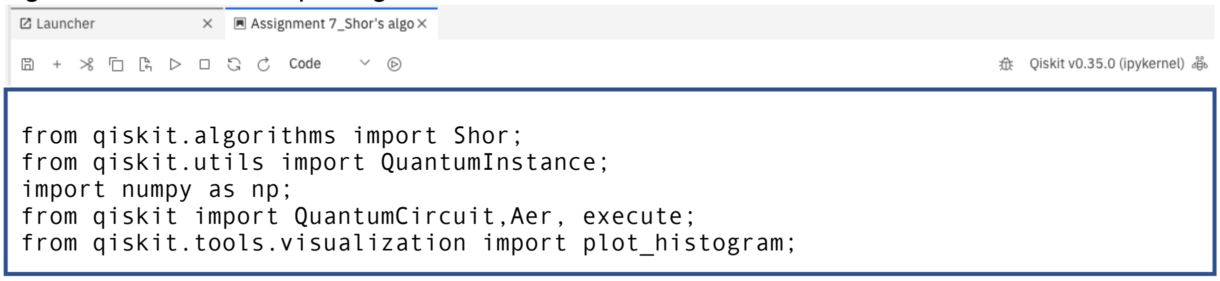
```
num = 21
prime factors(num)
```

We can see that the prime factors of the number 21 are 3 and 7.



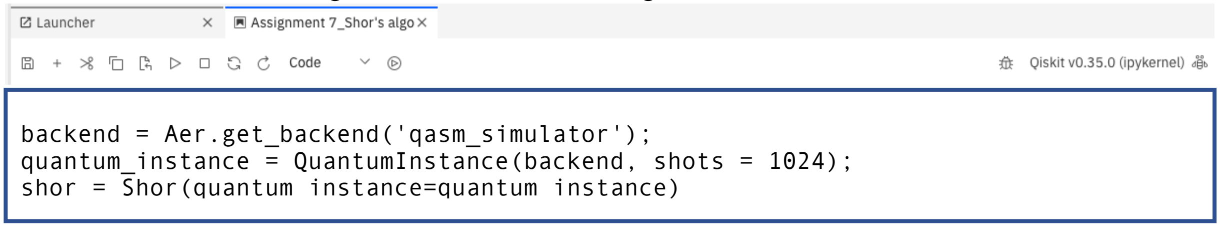
```
3
7.0
```

To accomplish factorization on a Quantum Computer, Qiskit allows us to use Shor's algorithm. However, due to the technological progress of Quantum Computers so far, we have a limited capacity of qubits and can hence only use the algorithm if the factorization requires less than 32 qubits. In our exemplary numbers above, only the factorization of 21 is possible with the following algorithm.
Again, we start with importing all relevant functions.



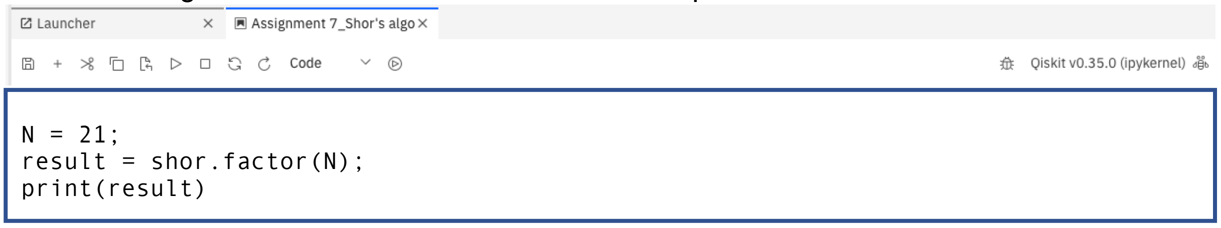
```
from qiskit.algorithms import Shor;
from qiskit.utils import QuantumInstance;
import numpy as np;
from qiskit import QuantumCircuit,Aer, execute;
from qiskit.tools.visualization import plot_histogram;
```

Please execute Shor's algorithm with the following code.



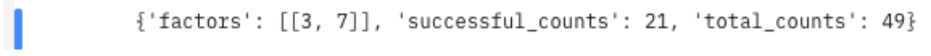
```
backend = Aer.get_backend('qasm_simulator');
quantum_instance = QuantumInstance(backend, shots = 1024);
shor = Shor(quantum_instance=quantum_instance)
```

Use Shor's algorithm to factorize the number 21 and print the result.



```
N = 21;
result = shor.factor(N);
print(result)
```

Again, we can see that the number 21 consists of 3 and 7.



```
{'factors': [[3, 7]], 'successful_counts': 21, 'total_counts': 49}
```

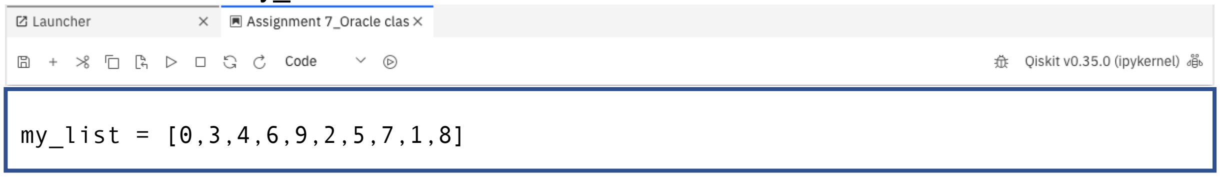
The larger the numbers become the longer it will take the Classical Computer to solve the prime factorization, while Quantum Computers are expected to solve the same problem in less time.

2. Grover's algorithm

The Indian scientist Lov Kumar Grover discovered the Grover's searching algorithm in 1996. His algorithm is used to search unstructured databases way faster than Classical Computers [2].
The Grover's algorithm can also be adapted to solve optimization problems [3], and hence can contribute to fight some of the biggest global challenges [4]. Please watch the following video to understand how quantum computing and optimization algorithms like Grover's can be used to create a more sustainable future: https://youtu.be/JUG9MXB7UK4

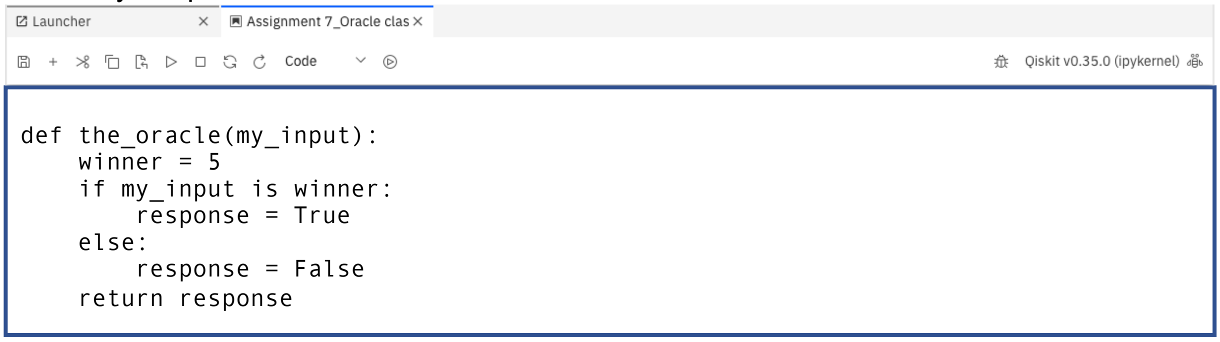
To understand the computational advantages of Grover's algorithm imagine you have a list of numbers 0,3,4,6,9,2,5,7,1,8 and you want to figure out at which position you can find number 5. A Classical Computer would go through each of the numbers in the list and check whether the number is equal to 5.

We can implement this example in the Quantum Lab. At first, we assign the list of numbers to a variable called “my_list”.



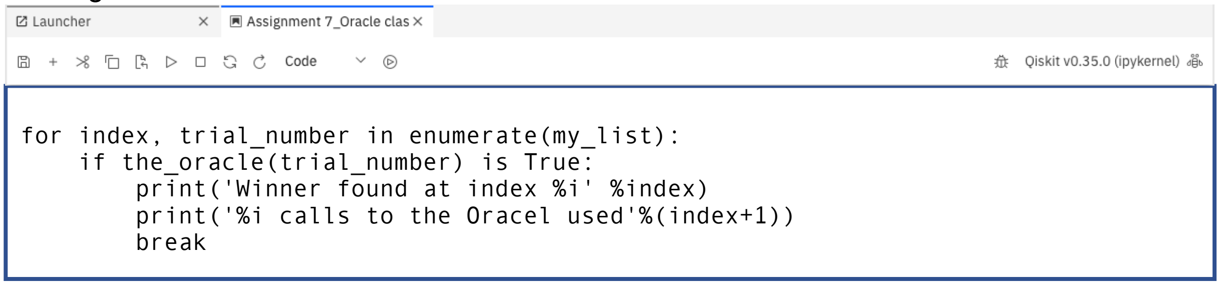
```
my_list = [0,3,4,6,9,2,5,7,1,8]
```

In the next step we define a function called “the_oracle” that responds “True” if we were able to identify the position of number “5” in the list and “False” otherwise.



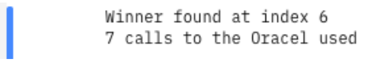
```
def the_oracle(my_input):
    winner = 5
    if my_input is winner:
        response = True
    else:
        response = False
    return response
```

The next part of the code tells us how many tries the Classical Computer would need to identify the target number 5.



```
for index, trial_number in enumerate(my_list):
    if the_oracle(trial_number) is True:
        print('Winner found at index %i' %index)
        print('%i calls to the Oracel used'%(index+1))
        break
```

In this example a Classical Computer would need 7 tries to identify the number 5 in the list.



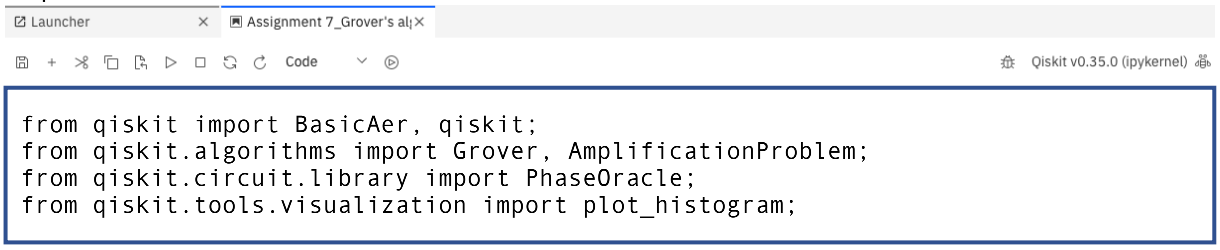
```
Winner found at index 6
7 calls to the Oracel used
```

You can also try to change the position of number 5 in the list and re-run the code. You will notice that, in some cases when the number 5 is either closer to the beginning of the list, less tries are required to find number 5 in the list. However, when number 5 is closer to the end of the list, it takes more trials to identify the position of the target. In general, we can conclude that on average a Classical Computer needs N/2 tries to search an unstructured database with N entries. By making use of quantum mechanics like superposition, Grover’s algorithm is able to reduce the required tries to search databases from N/2 to $\sqrt{N}$ [2].

In the next exercise you will practically learn how Grover’s algorithm can be used to search databases.

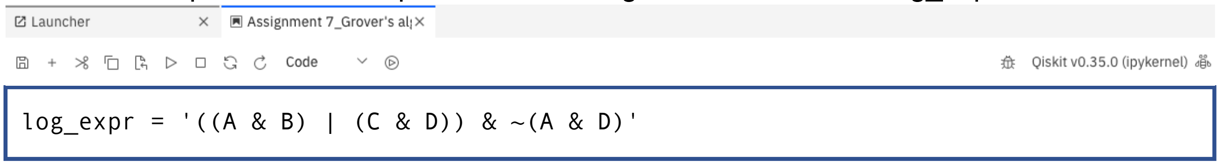
In our example we are going to use Grover’s search algorithm provided by Qiskit to decide who is going to be invited to our party. Imagine you have 4 friends, Alice and Bob as well as Carol and David, who are both in a relationship. However, as Alice and David had a bad breakup a while ago, they don’t get along with each other. By using Grover’s algorithm, you want to identify which combinations of guests can be invited without risking a bad atmosphere at your party. This kind of problem you want to solve is called a Boolean Satisfiability problem (SAT) and can be expressed as follows: ((A and B) or (C and D)) and not (A and D).

To be able to apply Grover’s algorithm on the aforementioned problem we need to import the required functions.



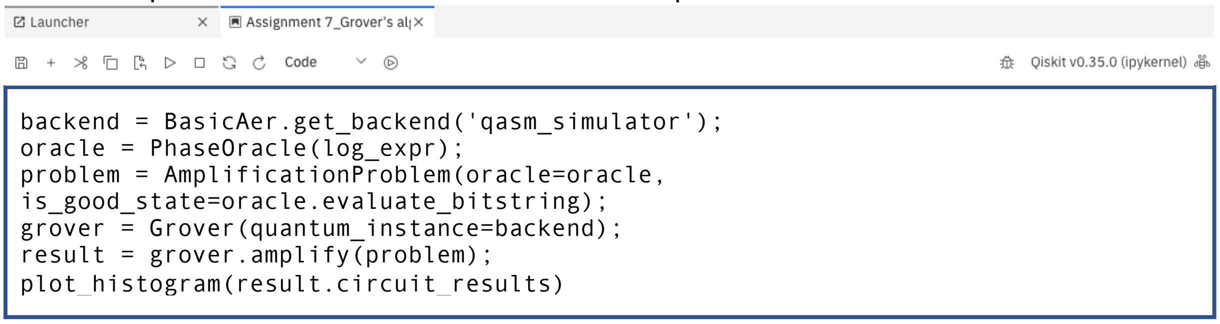
```
from qiskit import BasicAer, qiskit;
from qiskit.algorithms import Grover, AmplificationProblem;
from qiskit.circuit.library import PhaseOracle;
from qiskit.tools.visualization import plot_histogram;
```

In the next step we define our problem and assign it to the variable “log_expr”.



```
log_expr = '((A & B) | (C & D)) & ~(A & D)'
```

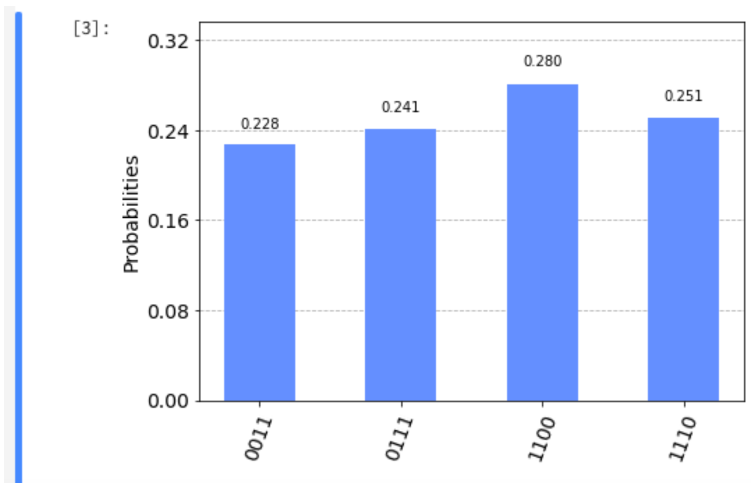
We then create our oracle and use Grover’s algorithm to solve it. At the end of the code, we extract the possible combinations of invitations and plot them.



```
backend = BasicAer.get_backend('qasm_simulator');
oracle = PhaseOracle(log_expr);
problem = AmplificationProblem(oracle=oracle,
is_good_state=oracle.evaluate_bitstring);
grover = Grover(quantum_instance=backend);
result = grover.amplify(problem);
plot_histogram(result.circuit_results)
```

As we have to read the string of qubits from right to left, we can conclude that Grover’s algorithm suggests the following combinations:

- Alice and Bob
- Alice, Bob and Carol
- Carol and David
- Bob, Carol and David



Note: Please don’t get irritated when you observe different probabilities of the potential combinations in your example. This is according to our expectations as we are summarizing several simulations in our experiment and each simulation can lead to a different outcome.

With a Classical Computer it would have taken us several trials to come up with the potential combinations that would risk a hostile atmosphere at our party, while Grover’s algorithm was able to solve the problem with only one try.

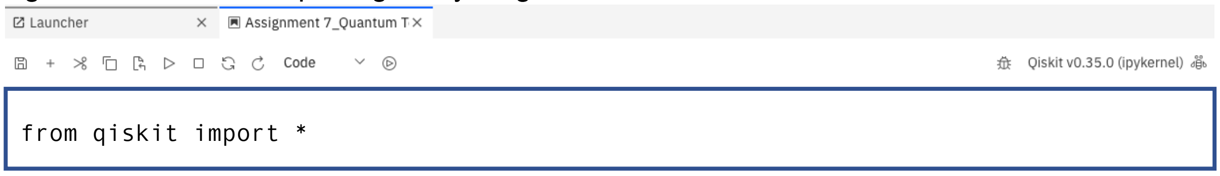
3. Quantum Teleportation

"Beam me up Scotty"
While teleportation is commonly portrayed in science fiction as a means to transfer humans or physical objects from one location to the next, quantum teleportation only transfers information. In quantum teleportation the state of a third particle is "teleported" to two distant, but entangled particles. Since 2019 it is confirmed that information can be passed between photons of computer chips even though the photons were not physically linked to each other [5]. The longest distance of successful teleportation which was achieved by the group of Jian-Wei Pan using the Micius satellite for space-based quantum teleportation was 1,400 km [6].

You may be wondering why we need quantum teleportation?
In traditional computing we copy and paste information easily without problems. However, in quantum computing the act of trying to transfer information by copying is not allowed because the moment you copy, you're actually doing a measurement which destroys the quantum state. In order to avoid this problem, we are going to take advantage of entanglement as a resource and build a circuit which is the quantum teleportation circuit [7].
How can quantum teleportation be used in real life?
Especially in the context of quantum internet and secure communication quantum teleportation will play a crucial role. Please watch the following video to get more information: https://youtu.be/vBjxI-5Fb4U

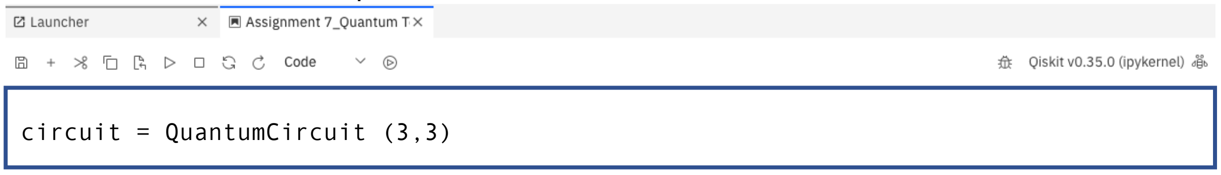
Let's check how we can teleport information by implementing a teleportation algorithm.

Again, we start with importing everything from Qiskit.



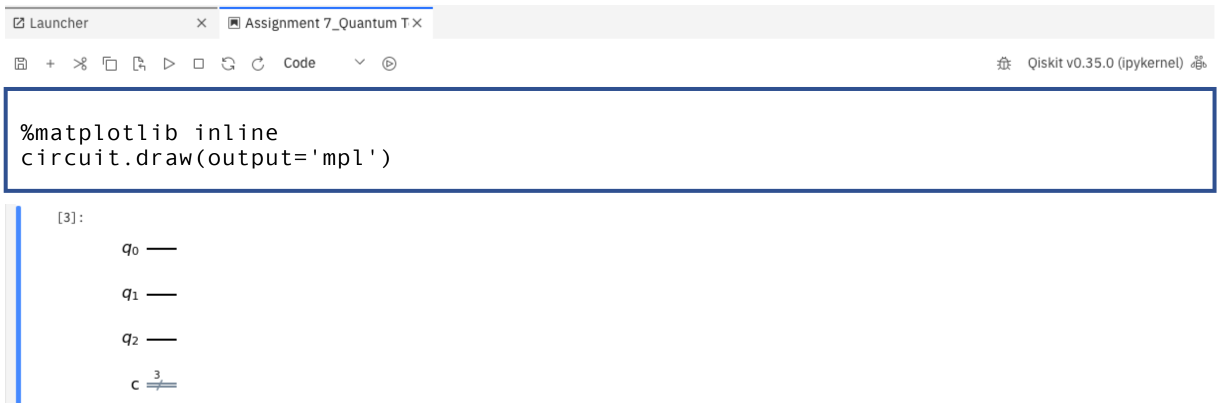
```
from qiskit import *
```

Create a circuit with three qubits and three bits.



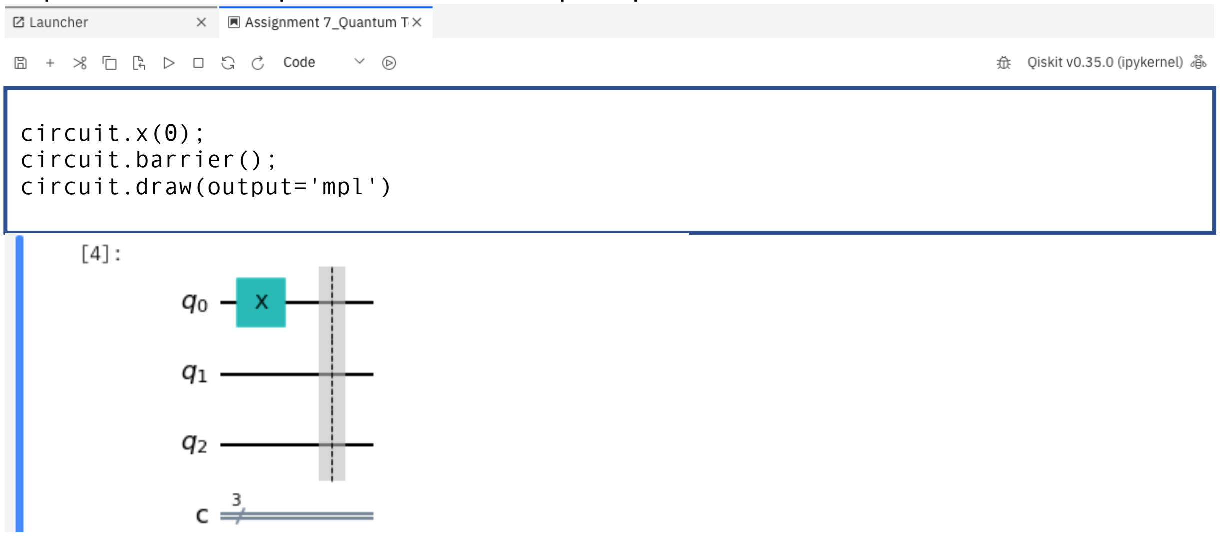
```
circuit = QuantumCircuit (3,3)
```

We are going to draw the circuit every step and want to see how the manipulations change the circuit.



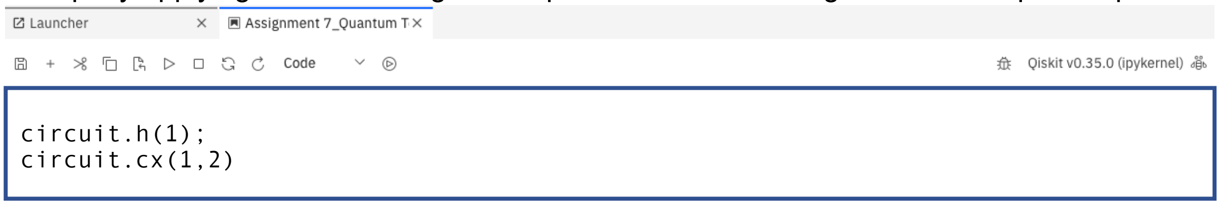
```
%matplotlib inline
circuit.draw(output='mpl')
```

In this step we start editing the teleportation circuit by applying a X-gate on q0. Additionally, we add a barrier after the first operation and draw the output. By executing the X-gate, we flipped the state of q0 right after the barrier from state 0 (initial state) to state 1. In the next steps we want to teleport the state 1 from q0 to q2.



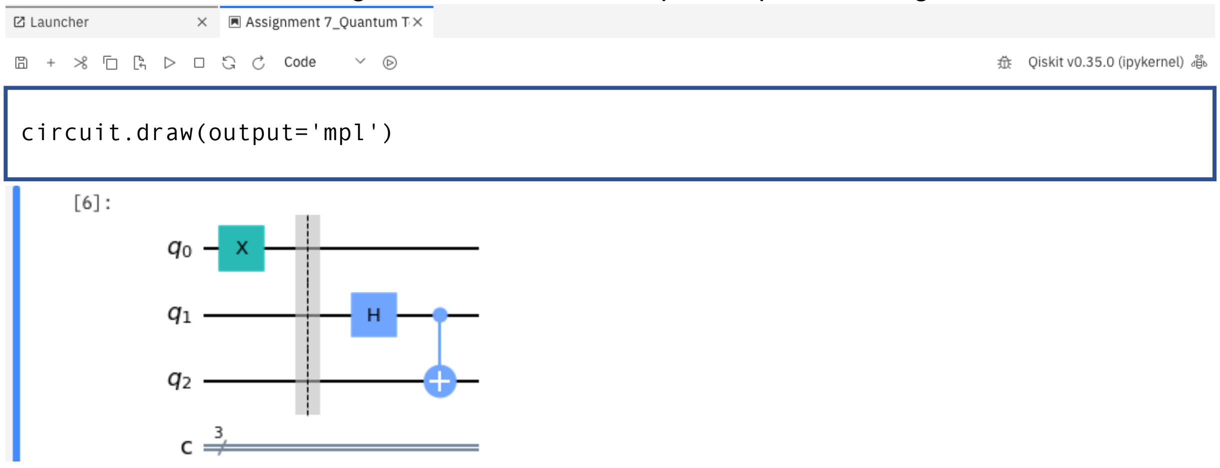
```
circuit.x(0);
circuit.barrier();
circuit.draw(output='mpl')
```

In the first step of the teleportation protocol, we are going to create entanglement between q1 and q2 by applying a Hadamard-gate on q1 and a controlled X-gate between q1 and q2.



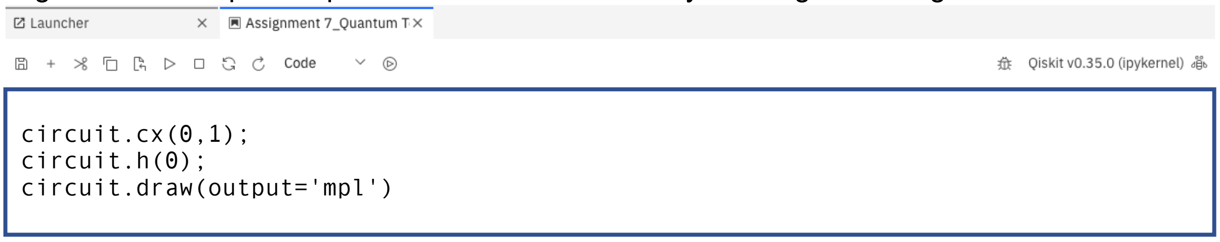
```
circuit.h(1);
circuit.cx(1,2)
```

We draw the circuit once again. You can see that q1 and q2 are entangled.



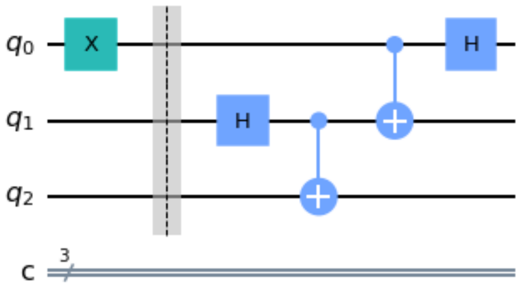
```
circuit.draw(output='mpl')
```

We continue the teleportation protocol by applying a Hadamard-gate on q0 and a controlled X-gate between q0 and q1 and visualize the circuit by drawing it once again.

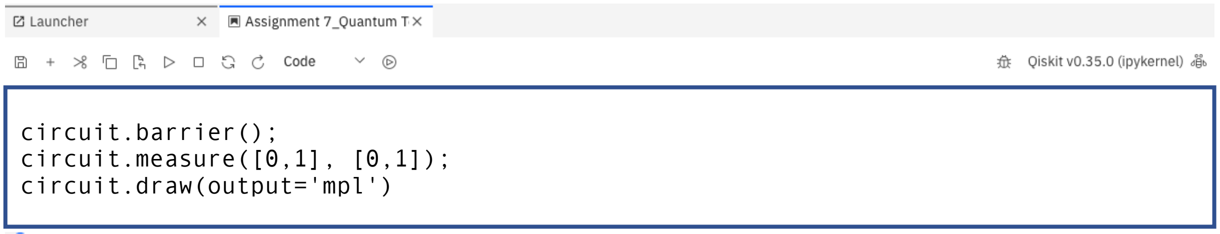


```
circuit.cx(0,1);
circuit.h(0);
circuit.draw(output='mpl')
```

[7]:

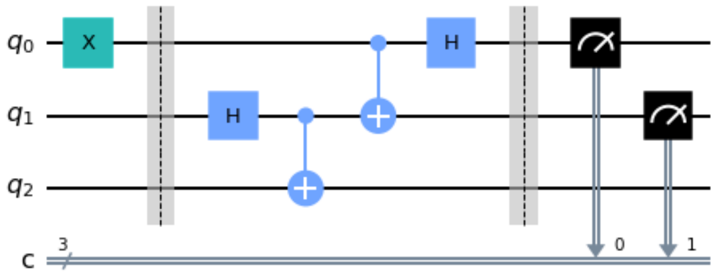


Now we want to do some measurements of q0 and q1. To execute the measurement, please use the code below.

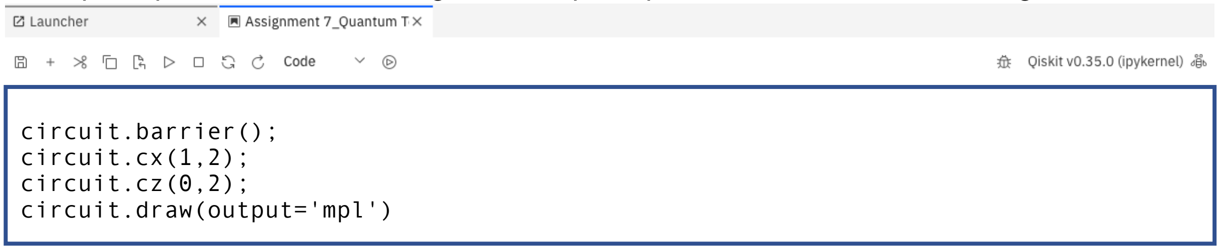


```
circuit.barrier();
circuit.measure([0,1], [0,1]);
circuit.draw(output='mpl')
```

[8]:

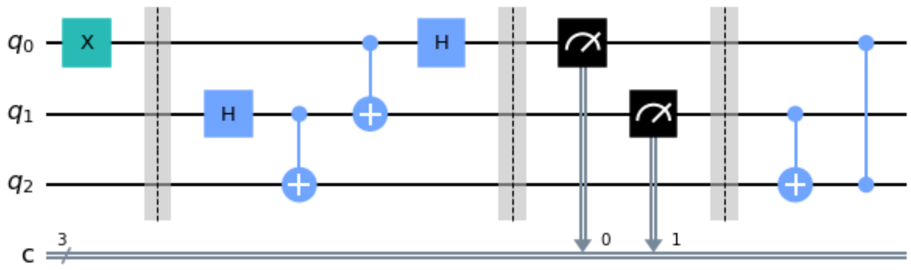


To complete our teleportation, we need to use two more gates. We apply a controlled X-gate from q1 to q2 and a controlled Z-gate from q0 to q2 and draw the final circuit again.

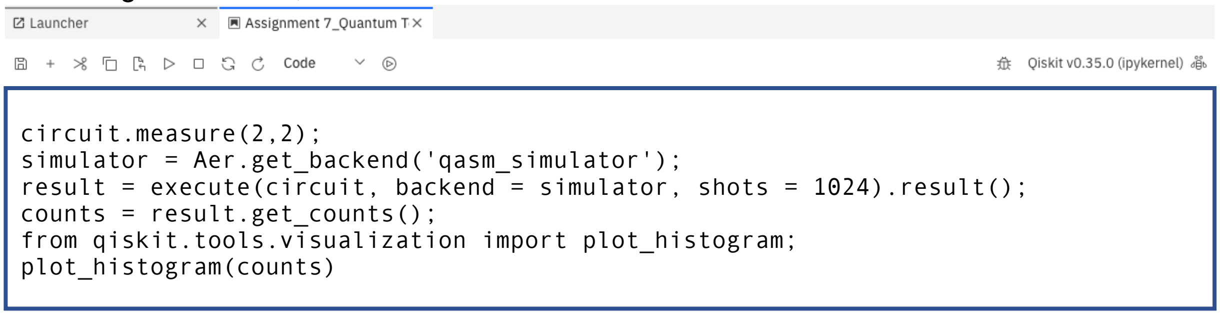


```
circuit.barrier();
circuit.cx(1,2);
circuit.cz(0,2);
circuit.draw(output='mpl')
```

[9]:

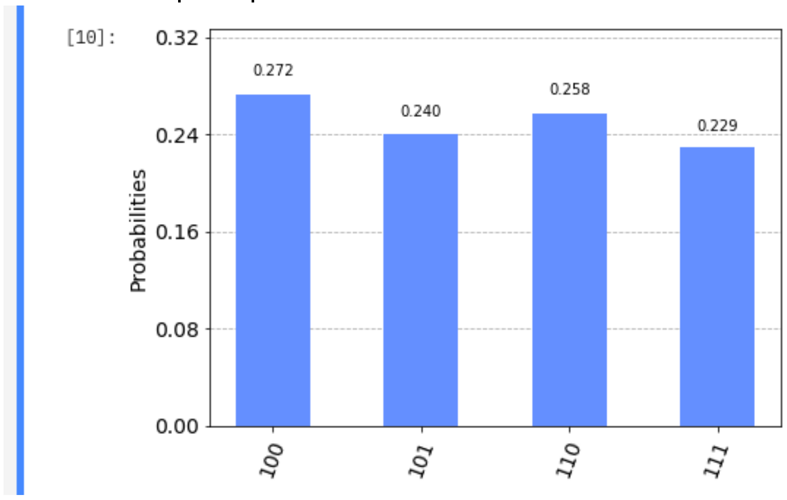


To verify if the teleportation of state 1 from q0 to q2 worked out, we can execute the circuit and measure the state of the second qubit. The last step of the code below draws the output of simulating the circuit 1,024 times.



```
circuit.measure(2,2);
simulator = Aer.get_backend('qasm_simulator');
result = execute(circuit, backend = simulator, shots = 1024).result();
counts = result.get_counts();
from qiskit.tools.visualization import plot_histogram;
plot_histogram(counts)
```

As you can see, the state of q2 is equal to 1 in all simulations indicating that teleporting the state 1 from q0 to q2 was successful.



You are also provided with an algorithm that enables a Quantum Computer to guess a secret number with one shot in the appendix of this assignment.

In case you still have plenty time left, please feel free to use the additional resources that are provided in the appendices of assignment 3 and 6, or download and play the Hello Quantum Game provided in assignment 3. Additionally, you can learn more about amplitudes of Quantum Computing in the optional assignment. You also have the chance to try out a real quantum device and error mitigation with the code corresponding to the video in assignment 5.

Useful Resources for Own Research

Shor's Algorithm: https://qiskit.org/textbook/ch-algorithms/shor.html

Academic GIFs: What is Post-Quantum Cryptography? And why do we need it? https://www.youtube.com/watch?v=wD_Fk3pu4Mg

Quantum Cryptography Explained: https://www.youtube.com/watch?v=UiJiXNEm-Go

Grover's Algorithm: https://qiskit.org/textbook/ch-algorithms/grover.html

Quantum Teleportation: https://qiskit.org/textbook/ch-algorithms/teleportation.html

Retrospective

Please answer the following questions:

1. What is the main task of Shor’s algorithm?
2. Why do we need Post Quantum Cryptography?
3. What is the main task of Grover’s algorithm?
4. How can Grover’s algorithm contribute to sustainability?
5. In which context do we need Quantum Teleportation?
6. Which requirements are needed to enable Quantum Teleportation?

Sources

[1] https://news.mit.edu/2016/quantum-computer-end-encryption-schemes-0303
[2] https://towardsdatascience.com/grovers-search-algorithm-simplified-4d4266bae29e
[3] https://research-management.mq.edu.au/ws/portalfiles/portal/62430609/Publisher+version+%28open+access%29.pdf
[4] https://arxiv.org/pdf/1711.07825.pdf
[5] https://www.nsf.gov/discoveries/disc_summ.jsp?cntn_id=300854&org=NSF&from=news
[6] https://www.nature.com/articles/nature23675/
[7] https://www.youtube.com/watch?v=mMwovHK2NrE
[8] https://learn.qiskit.org/course/introduction/why-quantum-computing#why-2-0

Appendix

Secret Number
In our last algorithm we want to uncover a secret number in a box with one shot.

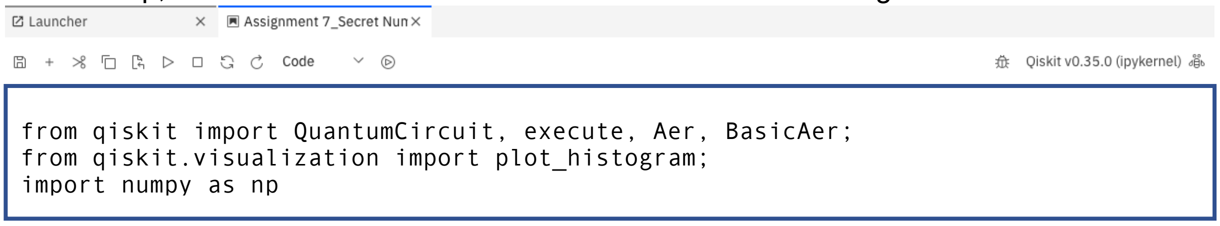
Let’s assume we have a 6 digits binary string of a hidden number sealed in a box.

A Classical Computer needs 6 tries to guess a 6 bit secret number, 60 tries to guess a 60 bit secret number and so on. In general, a Classical Computer needs n tries to guess a secret number with n digits.

How is it done? A Classical Computer would run as many tries as needed, starting from right to left until the first number is revealed. This process is called AND operation. When the rightmost digit is revealed, the algorithm repeats the algorithm for the other positions until the whole binary string is revealed. A Quantum Computer on the other hand, is able to determine the number in one shot by taking advantage of the Bernstein-Vazirani algorithm.

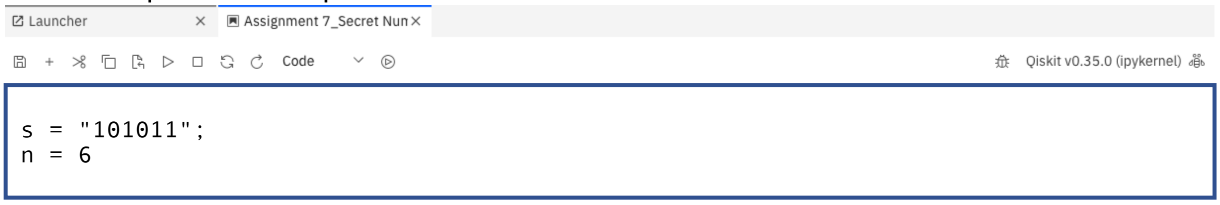
How does it works?

As first step, load some basic instruments before we execute our algorithm.



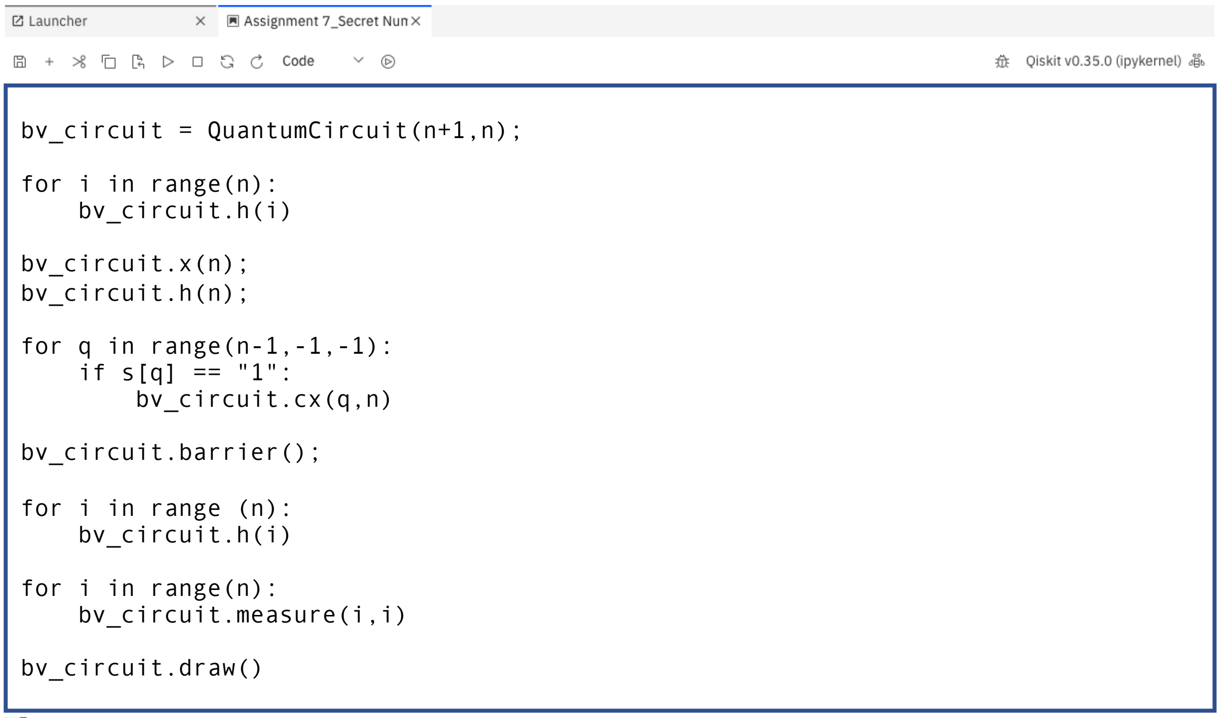
```
from qiskit import QuantumCircuit, execute, Aer, BasicAer;
from qiskit.visualization import plot_histogram;
import numpy as np
```

Now, let’s define our secret number as variable “s”. You can use any binary number you like. Please make sure to assign the number of qubits that are needed to represent your secret number to the variable “n”. In this case the secret number 101011 has six digits and therefore needs six qubits to be represented.



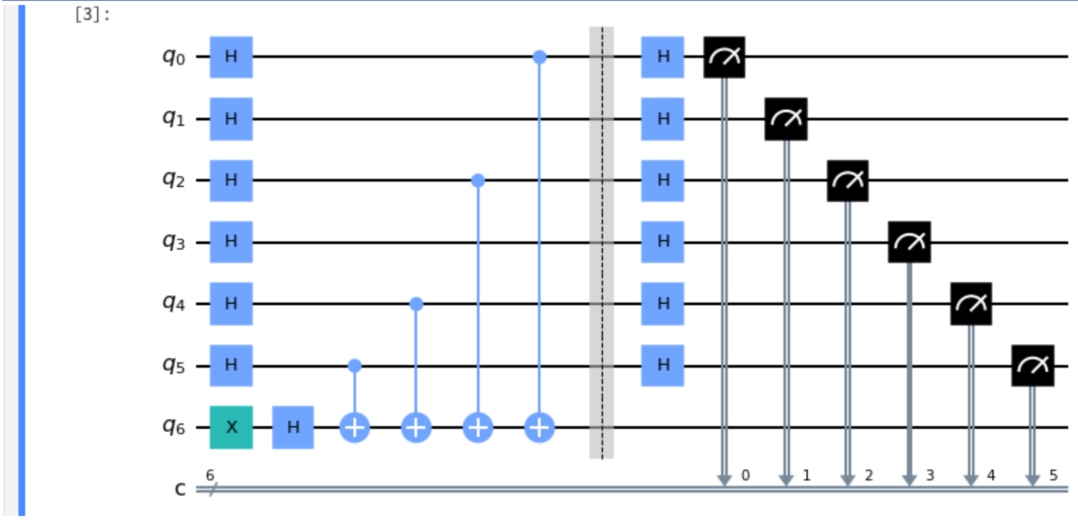
```
s = "101011";
n = 6
```

With the command bv_circuit = QuantumCircuit (n+1, n) we want to build our quantum circuit with 7 qubits in total (6 qubits + 1 additional qubit). As you know from the previous assignments, we always need the same number of classical bits to store the results.
In the next step a Hadamard-gate is applied to all qubits from q0 to q5. We then apply a X-gate and a Hadamard-gate on qubit n. The next for-loop in the code applies a controlled X-gate on all qubits in the secret number that are equal to 1. The target qubit of those controlled X-gates is always qubit n. The manipulations on the circuit are completed with applying a further Hadamard-gate on all qubits from q0 to q5. We then add a measurement for all qubits and draw the circuit.

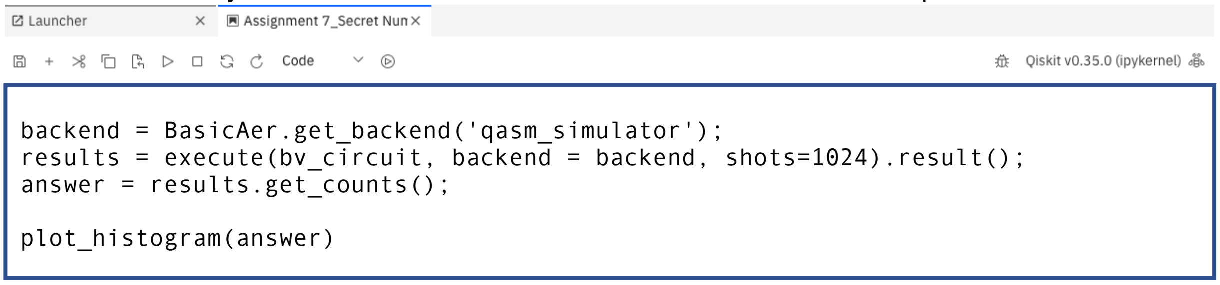


```
bv_circuit = QuantumCircuit(n+1,n);

for i in range(n):
    bv_circuit.h(i)

bv_circuit.x(n);
bv_circuit.h(n);

for q in range(n-1,-1,-1):
    if s[q] == "1":
        bv_circuit.cx(q,n)

bv_circuit.barrier();

for i in range (n):
    bv_circuit.h(i)

for i in range(n):
    bv_circuit.measure(i,i)

bv_circuit.draw()
```

[3]:

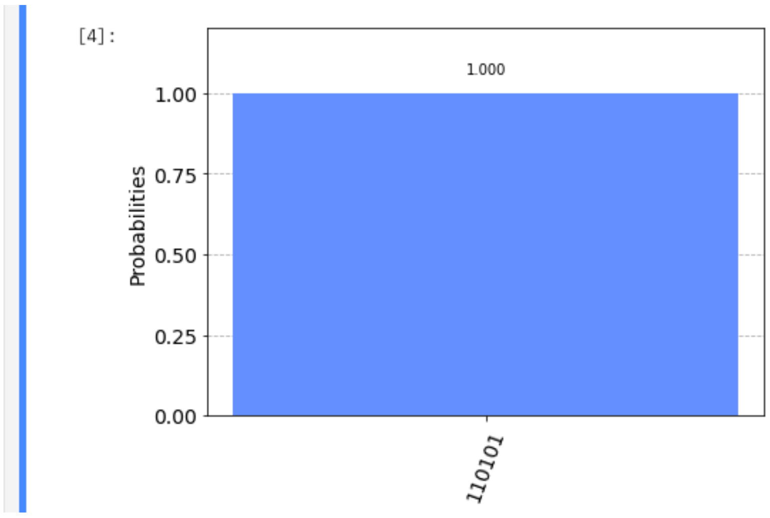


Now we are ready to simulate the circuit with the QASM simulator and print the outcome.



```
backend = BasicAer.get_backend('qasm_simulator');
results = execute(bv_circuit, backend = backend, shots=1024).result();
answer = results.get_counts();

plot_histogram(answer)
```

As we need to read strings of qubits from right to left, the outcome of the graph indicates that the secret number is 101011. This is in line with the secret number we defined in variable s.



You can also change the number of “s” and re-run the algorithm to check if it still works.

We can conclude that it took the Quantum Computer only one try to identify the correct secret number and hence the experiment was successful.