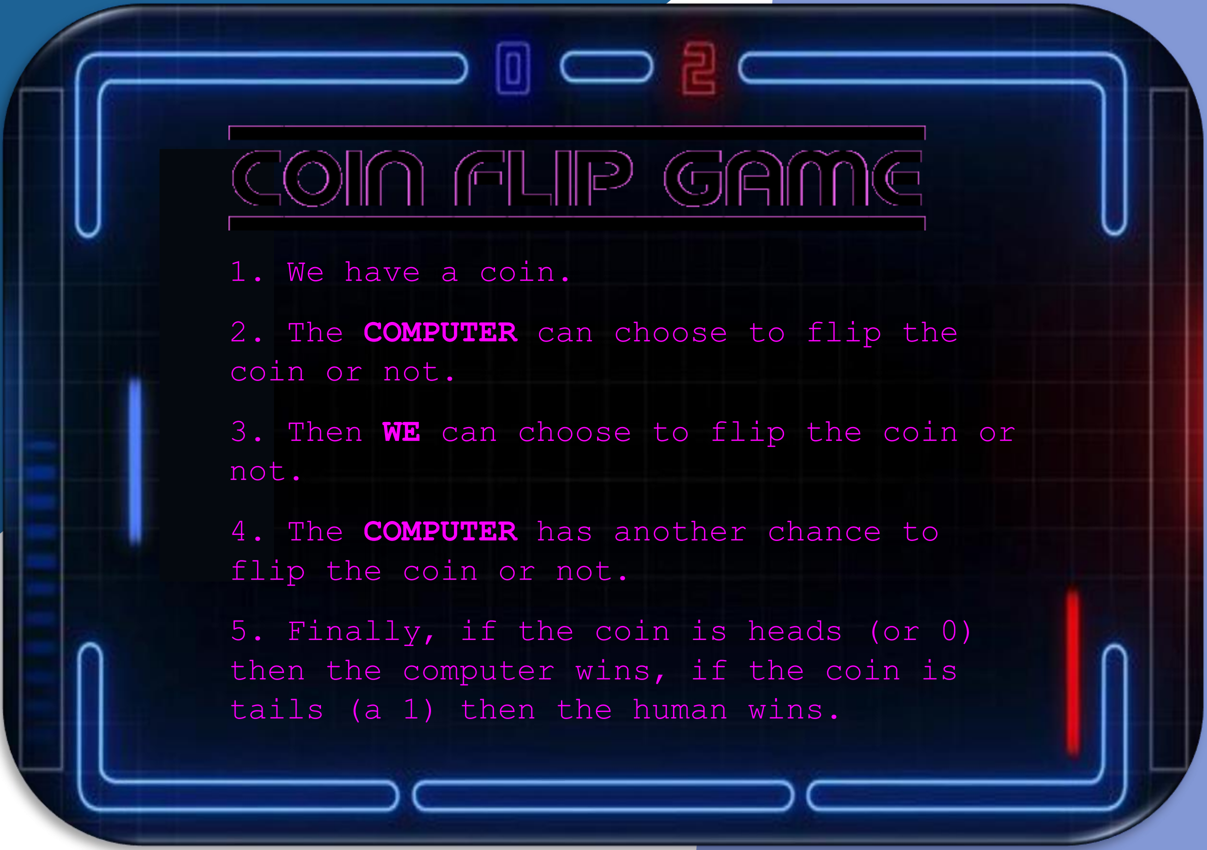
Quantum Coin Flip

Note: This tutorial uses a quantum computing simulator (pretend computer). Its possible to test this code on a real quantum computer if you regsiter an account with IBMQ.

We are going to play a coin flipping game against the computer. The game goes like this:



COIN FLIP GAME

1. We have a coin.

2. The **COMPUTER** can choose to flip the coin or not.

3. Then **WE** can choose to flip the coin or not.

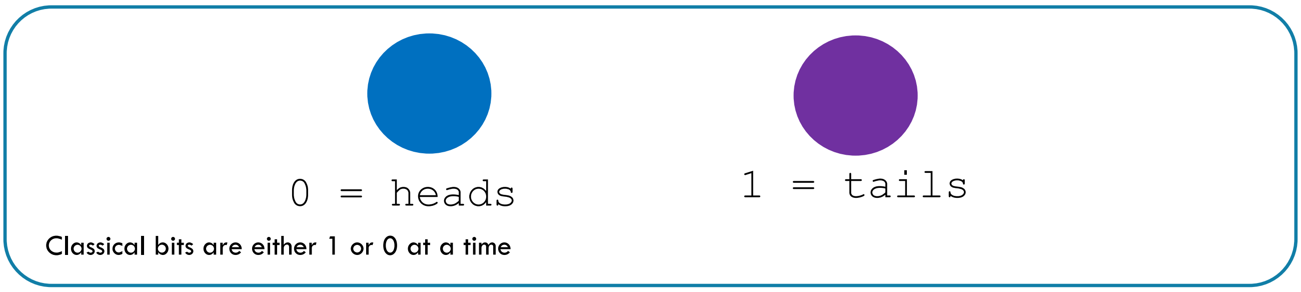
4. The **COMPUTER** has another chance to flip the coin or not.

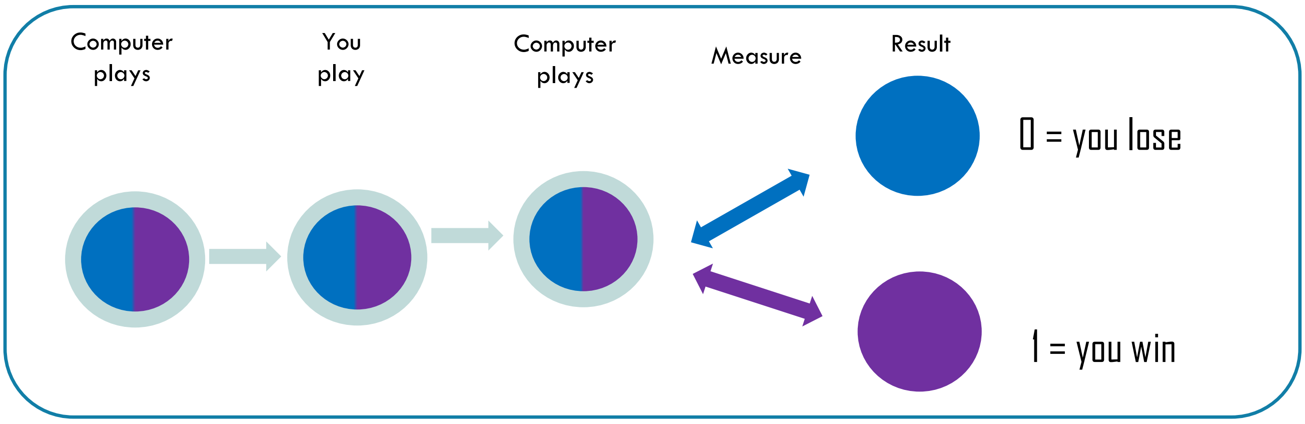
5. Finally, if the coin is heads (or 0) then the computer wins, if the coin is tails (a 1) then the human wins.

Under normal circumstances, when a coin is fair, we would have 50 % chance of winning the game (There are two sides of the coin)

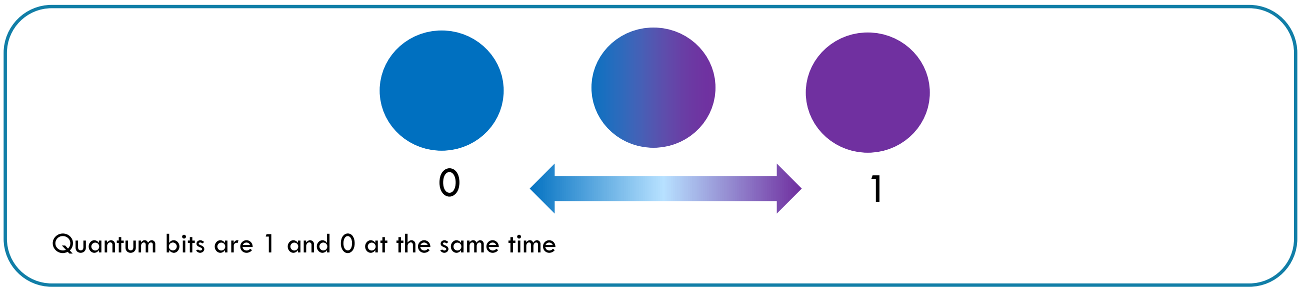
Against a quantum computer however, we will lose the game, 97% of the time. As described in this TED talk.

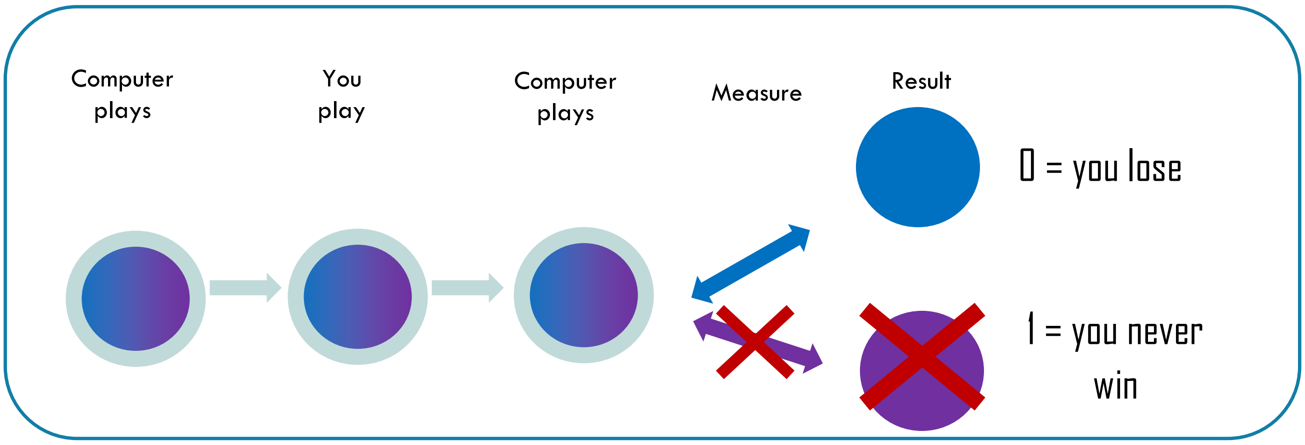
The reason for this is that a classical computer bit can be one of two things, a 1 or a 0. So the coin has a 50% chance of being a 1 or a 50% chance of being a zero. This means we have a 50% chance of winning.





A quantum bit (qubit) can be a 1 and a 0 at the same time (its identity lies on a spectrum as a combination/superposition of 1 and 2). So it has 100% chance of being a 1 and 100% chance of being a 0. Weird right? Only when we observe the qubit (measure) it, does it decide to be a 0 or a 1 so in the result, at the last second it can decide to be a 0. And it does decide to be a 0. Quantum computers don't always work well, so sometimes it doesn't work. But its very rare (3%)





Lets code this problem in Python!

We will code a coin flip game using classical computer knowledge and one with quantum computer knowledge. We will build it here https://repl.it/languages/python3

Step 1: To use the quantum computing simulator (fake computer) we have to import certain libraries. We also want to import the "random" library so we can test it on a classical computer:

```
import random
import qiskit as qk #quantum computing library
from qiskit import execute
from qiskit import Aer
```

Note: The first time you run the program it will download and install a bunch of stuff into the python console (specifically the qiskit library.) It will be done when it says

```
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Lets build the calssical computer flipping game first:

Step 2: lets define a `classical_coin_flip()` function and start our coin at heads (HEADS = 1).

```
def classical_coin_flip():
  #chooses to flip or not flip, it it choses to flip it flips
  coin = 1
```

Step 3: lets define two helper functions that will helps us build our classical coin flipping game. The first funciton we will define is `flip_coin()`. This function will randomly give us (return) a 0(heads) or a 0(tails). Just like flipping a coin. We use the `random.randint()` function. It will return random integers between the two numbers provided. In our case this is 0 and 1. So it will randomly return a heads (0) or a tails(1) value. The second helper function is a `computer_play()` function. First it helps the computer decide if it wants to flip the coin, then it flips the coin if necessary.

```
def flip_coin():
  return(random.randint(0,1)) #returns a 0 or 1 randomly

def computer_play(coin_var):
  #chooses to flip or not flip, it it choses to flip(1) it flips
  flip_or_not = random.randint(0,1)
  if flip_or_not == 1:
    return(flip_coin()) #it calls the flip function to flip the coin
  else:
    return coin_var
```

Step 4: The first step in the classical coin flip game is to let the computer play. We can do this by calling the `computer_play()` function we wrote and by passing this function our coin variable. Next we want to ask the user if they would like to flip the coin or not using input. If they do want to flip the coin then we should flip it for them. So lets write this into our classical coin flip function. After the human plays, the computer plays again.

```
def classical_coin_flip():
  coin = 1
  coin = computer_play(coin) #computer plays
  humen = input("Would you like to flip? Yes(y) or No(n): ") #human plays
  if humen == "y": #if the human says yes we need to flip the coin for them
    coin = flip_coin()
  coin = computer_play(coin) #computer plays again
```

Step 5: The final step in the classical coin flip game is to see who has won. If the coin is a 1 then the computer has won. If the coin is a 0 then the human has won. Call you function to play your game.

```
def classical_coin_flip():
  coin = 1
  coin = computer_play(coin) #computer plays
  humen = input("Would you like to flip? Yes(y) or No(n): ") #human plays
  if humen == "y": #if the human says yes we need to flip the coin for them
    coin = flip_coin()
  coin = computer_play(coin) #computer plays again

  if coin == 0:
    print("You lose. Soz")
  if coin == 1:
    print("You win! Well done")

classical_coin_flip()
```

Run your program and play a few rounds and see if you win!

Step 6: Next lets code the quantum coin flipping game in a `quantum_coin_flip()` function. First we need to create a simulator using the Aer backend from IBM. After that we need 1 quantum register (a place to keep our qubit) and a classical register (a place to keep a classical bit). We need a classical bit because when we measure the quantum bit we need a place to keep it. Finally we need, we need a quantum circiut which is the instructions for the quantum system (translating python language to quantum computer language).

```python
def quantum_coin_flip():
  backend = Aer.get_backend("qasm_simulator") # create fake quantum computer
  qr = qk.QuantumRegister(1) #create Quantum Register with 1 qubit
  cr = qk.ClassicalRegister(1) #create Classical register with 1 bit
  qc = qk.QuantumCircuit(qr,cr) # create a quantum circuit acting on the bits
```

Step 7: When we want the computer to flip a quantum "coin" or radomly change the quantum bit from a 0 to a 1 or a 1 to a 0 we use something called a hadamard gate. A hadamard gate randomizes the state of a qubit (similar to the random function for integers). It is like asking the cmputer if it wants to play, and flipping the coin at the same time. When a human "flips" the qubit/coin however, we use an x gate which is a direct flip of the qubit state. Again, however, we first want to ask the human if they would like to play, and only use the x gate if they do. Lets add this to our quantum coin flip function.

```python
def quantum_coin_flip():
  backend = Aer.get_backend("qasm_simulator") # create fake quantum computer
  qr = qk.QuantumRegister(1) #create Quantum Register with 1 qubit
  cr = qk.ClassicalRegister(1) #create Classical register with 1 bit
  qc = qk.QuantumCircuit(qr,cr) # create a quantum circuit acting on the bits

  qc.h(qr[0]) # computer plays with hadamrd gate
  humen = input("Would you like to flip? Yes(y) or No(n): ")
  if humen == "y":
    qc.x(qr[0]) # human plays with an X gate, a qubit flip operator
  qc.h(qr[0]) # computer plays again
```

Step 8: The last instruction we want the quantum ciciut to perform is measuring the qubit. Now that we have writtin the flip instructions for the quantum circiut (qc) we will write a command to execute the instructions in the quantum circuit using our simulator, and we will tell it to run once in a "job".

```python
def quantum_coin_flip():
  backend = Aer.get_backend("qasm_simulator") # create fake quantum computer
  qr = qk.QuantumRegister(1) #create Quantum Register with 1 qubit
  cr = qk.ClassicalRegister(1) #create Classical register with 1 bit
  qc = qk.QuantumCircuit(qr,cr) # create a quantum circuit acting on the bits

  qc.h(qr[0]) # computer plays with hadamrd gate
  humen = input("Would you like to flip? Yes(y) or No(n): ")
  if humen == "y":
    qc.x(qr[0]) # human plays with an X gate, a qubit flip operator
  qc.h(qr[0]) # computer plays again

  qc.measure(qr,cr)
  job = execute(qc, backend = backend, shots = 1)
```

Step 9: The result of the game are retuned in the job.results() and we can access the value of the coin through the counts key where there was a count of how many times we get a 0. If the counts of how many times we got a 0 is 1 then we lose, if not then we win.

```
def quantum_coin_flip():
  backend = Aer.get_backend("qasm_simulator") # create fake quantum
computer
  qr = qk.QuantumRegister(1) #create Quantum Register with 1 qubit
  cr = qk.ClassicalRegister(1) #create Classical register with 1 bit
  qc = qk.QuantumCircuit(qr,cr) # create a quantum circuit acting on the
bits
  qc.h(qr[0]) # computer plays with hadamrd gate
  humen = input("Would you like to flip? Yes(y) or No(n): ")
  if humen == "y":
    qc.x(qr[0]) # human plays with an X gate, a qubit flip operator
  qc.h(qr[0]) # computer plays again
  qc.measure(qr,cr)
  job = execute(qc, backend = backend, shots = 1)

  results = job.result()

  counts = results.get_counts()

  if counts["0"] == 1:
    print("You lose. Soz")
  if counts["0"] == 0:
    print("You win, but it probably a mistake in the computer")
```

Step 10: The final step is to allow the user t choose which game they would like to play. We do this with a simple if statement.

```
game = input("Classical game (c) or Quantum game (q)?: ")
if game == "c":
  classical_coin_flip()
if game == "q":
  quantum_coin_flip()
```

Well done! You have made a game on a quantum computer! Not many people can say that! Your final code should like like below

```python
import random
import qiskit as qk
from qiskit import execute
from qiskit import Aer

def quantum_coin_flip():
  backend = Aer.get_backend("qasm_simulator")
  qr = qk.QuantumRegister(1)
  cr = qk.ClassicalRegister(1)
  qc = qk.QuantumCircuit(qr,cr)

qc.h(qr[0])
  humen = input("Would you like to flip? Yes(y) or No(n): ")
  if humen == "y":
    qc.x(qr[0])
  qc.h(qr[0])

  qc.measure(qr,cr)
  job = execute(qc, backend = backend, shots = 1)

  results = job.result()
  counts = results.get_counts()
  if counts["0"] == 1:
    print("You lose. Soz")
  if counts["0"] == 0:
    print("You win, but it probably a mistake in the computer")


def flip_coin():
  return(random.randint(0,1))

def computer_play(coin_var):
  flip_or_not = random.randint(0,1)
  if flip_or_not == 1:
    return(flip_coin()) #it calls the flip function to flip the coin
  else:
    return coin_var

def classical_coin_flip():
  coin = 1
  coin = computer_play(coin)
  humen = input("Would you like to flip? Yes(y) or No(n): ")
  if humen == "y":
    coin = flip_coin()
  coin = computer_play(coin)
  print(coin)
  if coin == 0:
    print("You lose. Soz")
  if coin == 1:
    print("You win! Well done")

game = input("Classical game (c) or Quantum game (q)?: ")
if game == "c":
  classical_coin_flip()
if game == "q":
  quantum_coin_flip()
```