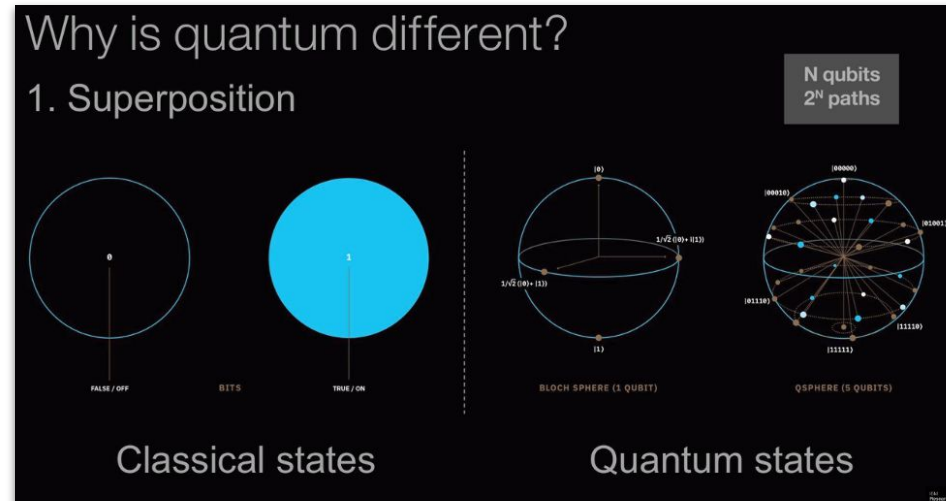

Quasm Calculator

How we implemented rigorous software testing methods into a homemade piece of quantum computing software

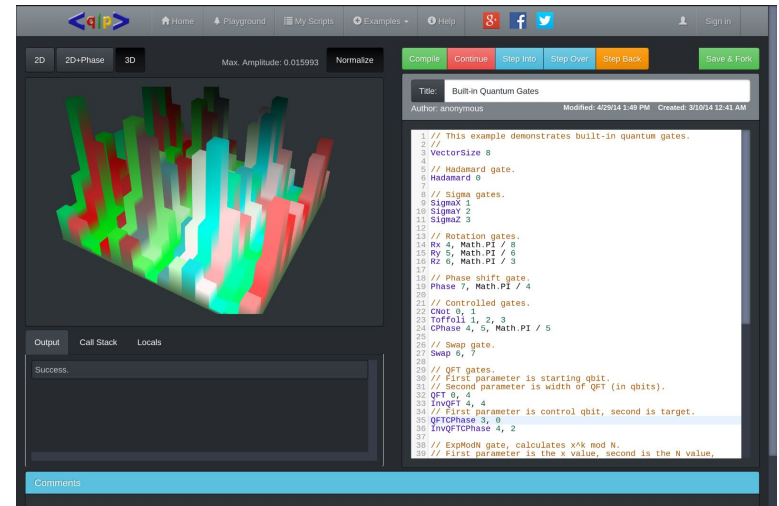
What is quantum computing?

Quantum computing is the use of quantum phenomena such as superposition and entanglement to perform computation. Computers that perform quantum computations are known as quantum computers.



What is a quantum simulator?

Quantum simulators are software programs that run on classical computers and make it possible to run and test quantum programs in an environment that predicts how qubits react to different operations.



The screenshot shows a web-based quantum simulator interface. The top navigation bar includes links for Home, Playground, My Scripts, Examples, Help, and social media icons. The main interface is divided into several sections:

- Top Bar:** Contains navigation buttons (2D, 2D+Phase, 3D) and a 'Normalize' button. The 'Max. Amplitude' is displayed as 0.015993.
- 3D Visualization:** A 3D plot showing a complex, multi-colored structure representing a quantum state in a high-dimensional space.
- Code Editor:** A text area containing quantum circuit code. The code defines various gates and operations, including Hadamard, Sigma, Rotation (Rx, Ry, Rz), Phase shift, controlled gates (CNOT, Toffoli), and Swap gates. Comments explain the parameters for each gate.
- Output Panel:** Located at the bottom left, it shows the result of the simulation: 'Success.'
- Metadata:** On the right side, it displays the title 'Built-in Quantum Gates', author 'anonymous', and modification/creation dates.

2.

Difference between ours and a full-scale simulator

IBM Quantum Simulator

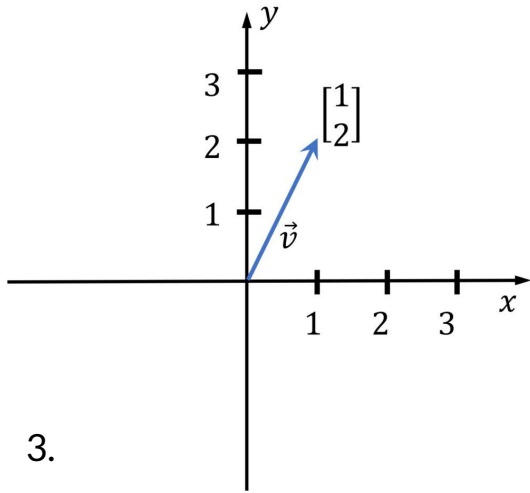
- Infinite number of states per finite number of qubits
- Custom number of qubits
- Custom unitary operators
 - Pauli-XYZ
 - Hadamard
 - Rotators
 - Controlled-Not
 - Swap



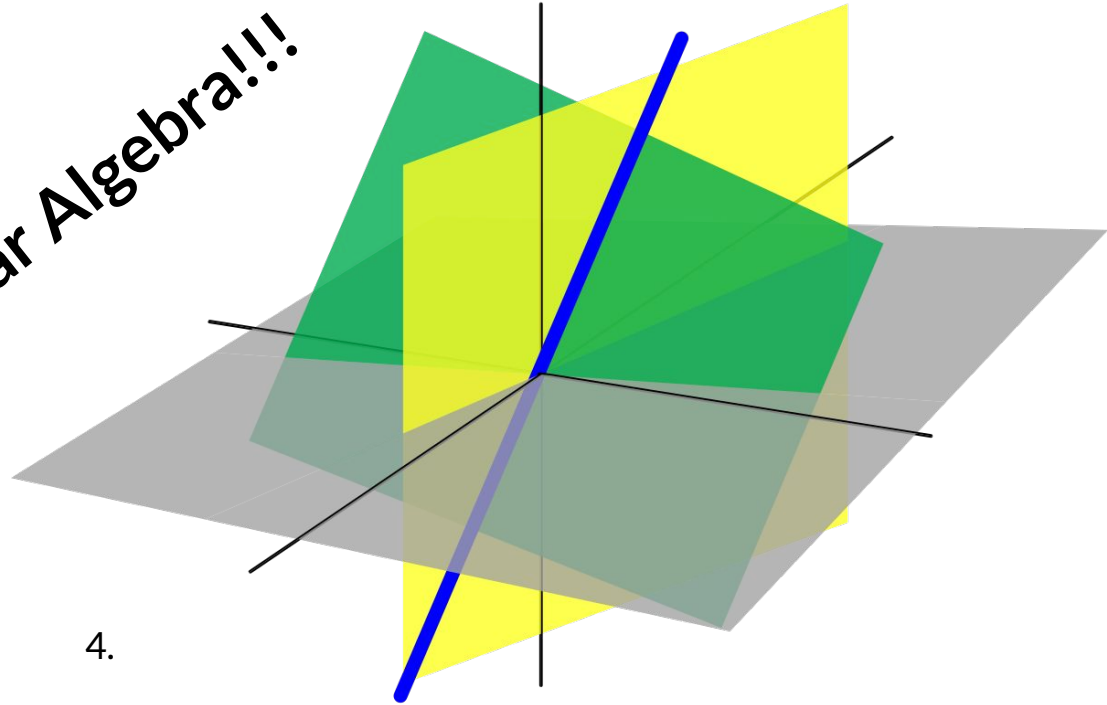
Our Quantum Simulator

- Finite number of states per finite number of qubits
- 2 set qubits
- Finite operations
 - X
 - Hadamard
 - Controlled-Not
 - Swap

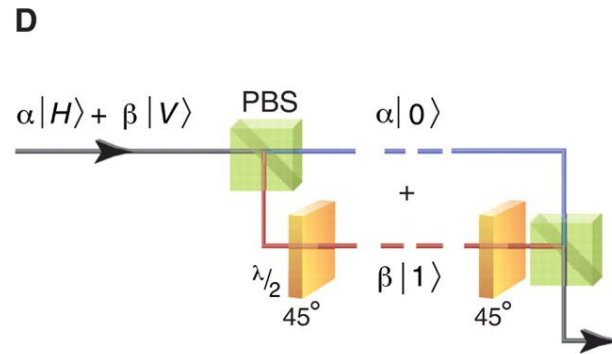
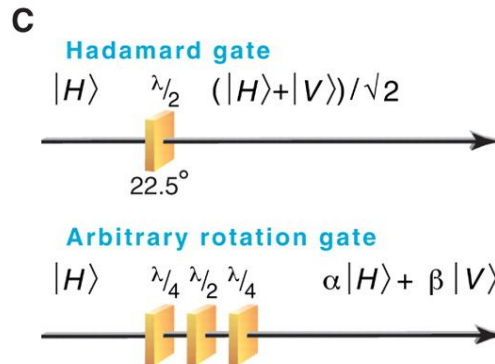
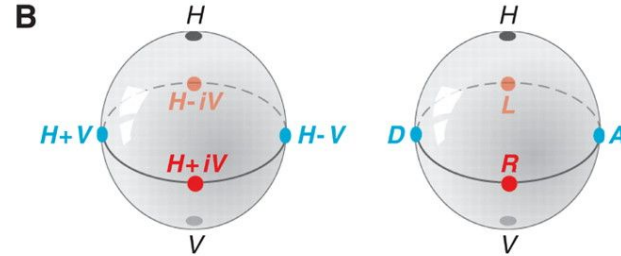
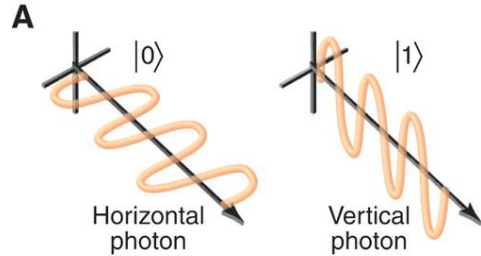
How does the simulator work?



Linear Algebra!!!



What it looks like under the hood



How the math works

$$\begin{aligned} |0\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} & H &= \sqrt{\frac{1}{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} & I &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ |1\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} & X &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & |00\rangle &= |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \end{aligned}$$

$$(H \otimes I) \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \sqrt{\frac{1}{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \sqrt{\frac{1}{2}} \\ \sqrt{\frac{1}{2}} \\ 0 \\ 0 \end{pmatrix}$$

How the math works

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad |01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad |11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = \begin{pmatrix} \sqrt{\frac{1}{2}} \\ 0 \\ 0 \\ \sqrt{\frac{1}{2}} \end{pmatrix}$$

$$\frac{|00\rangle + |10\rangle + |01\rangle + |11\rangle}{\sqrt{4}} = \begin{pmatrix} \sqrt{\frac{1}{4}} \\ \sqrt{\frac{1}{4}} \\ \sqrt{\frac{1}{4}} \\ \sqrt{\frac{1}{4}} \end{pmatrix}$$

Symbols

Symbol	Value
H	$\sqrt{\frac{1}{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
PAULI-X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
CNOT	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
SWAP	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
M	$\langle \Psi M^\dagger M \Psi \rangle$
T	Toggle

Symbol	Value
0	0
1	1
D	-1
R	$\sqrt{\frac{1}{2}}$
L	$-\sqrt{\frac{1}{2}}$
2	$\frac{1}{2}$
N	$-\frac{1}{2}$

Operations

Symbol	Target 0
H	$H \otimes I$
X	$X \otimes I$
C	$\downarrow CNOT$
S	$SWAP$
T	$TOGGLE$
M	$\langle \Psi M^\dagger M \Psi \rangle$

$$R \quad \begin{aligned} STATE &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\ TARGET &= 0 \end{aligned}$$

Symbol	Target 1
H	$I \otimes H$
X	$I \otimes H$
C	$\uparrow CNOT$
S	$SWAP$
T	$TOGGLE$
M	$\langle \Psi M^\dagger M \Psi \rangle$

$$R \quad \begin{aligned} STATE &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\ TARGET &= 0 \end{aligned}$$

Symbol	Value
0	0
1	1
D	-1
R	$\sqrt{\frac{1}{2}}$
L	$-\sqrt{\frac{1}{2}}$
2	$\frac{1}{2}$
N	$-\frac{1}{2}$

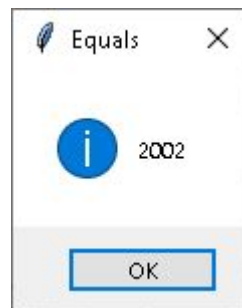
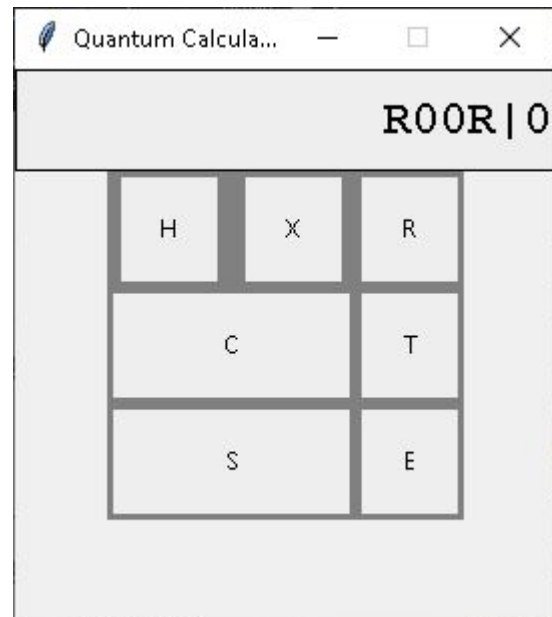
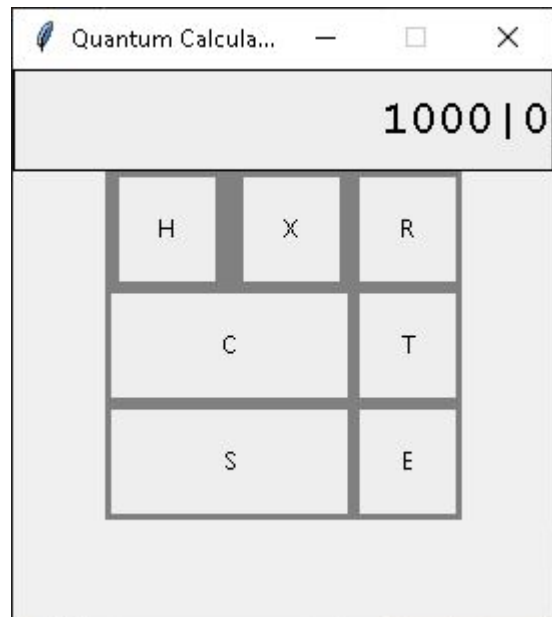
Our Application

```
(1000|0) [ X, H, T, R, C, S, E, ]> H
(RR00|0) [ X, H, T, R, C, S, E, ]> T
(RR00|1) [ X, H, T, R, C, S, E, ]> H
(2222|1) [ X, H, T, R, C, S, E, ]> H
(RR00|1) [ X, H, T, R, C, S, E, ]> T
(RR00|0) [ X, H, T, R, C, S, E, ]> C
(R00R|0) [ X, H, T, R, C, S, E, ]> T
(R00R|1) [ X, H, T, R, C, S, E, ]> X
(ORR0|1) [ X, H, T, R, C, S, E, ]> R
(1000|0) [ X, H, T, R, C, S, E, ]> H
(RR00|0) [ X, H, T, R, C, S, E, ]> T
(RR00|1) [ X, H, T, R, C, S, E, ]> E
```

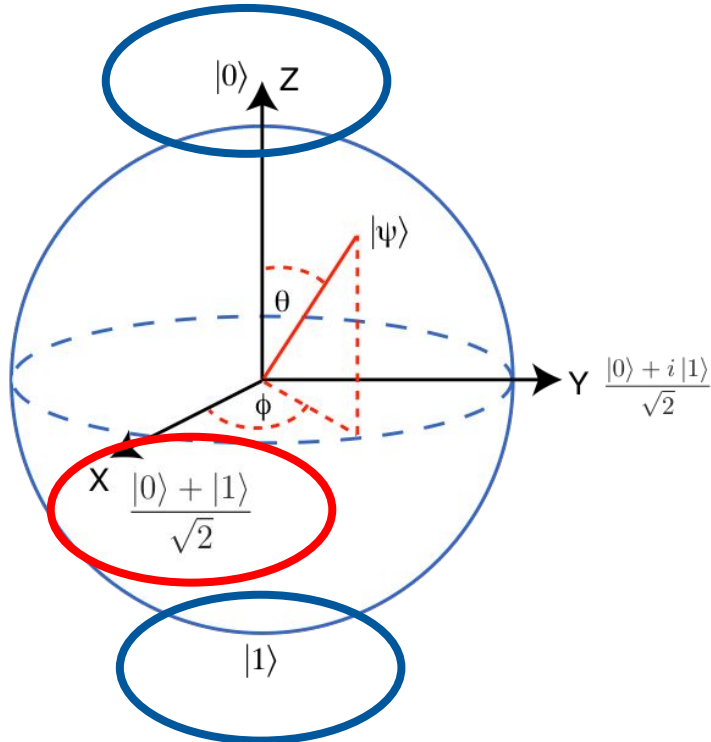
The final output is [2200|0]

Symbol	Value
0	0
1	1
D	-1
R	$\sqrt{\frac{1}{2}}$
L	$-\sqrt{\frac{1}{2}}$
2	$\frac{1}{2}$
N	$-\frac{1}{2}$

Our Application



Inner Working of Application



6.

```
class Calculator():
    t = 0
    s = [[1], [0], [0], [0]]
```

C = [

```
[
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 1],
    [0, 0, 1, 0]
], [
    [1, 0, 0, 0],
    [0, 0, 0, 1],
    [0, 0, 1, 0],
    [0, 1, 0, 0]
]
```

]

Linear Software Model

		Structors					
		Single Qbit Calculator	Multi Qbit Calculator	Transmutor	Reader	GUI	Alert
F u n c t i o n a l s	Toggle (T)	1	0	0	0	0	0
	Calculate (C)	1	1	0	0	0	0
	Transmute	0	0	1	0	0	0
	Read	0	0	0	1	0	0
	Draw	0	0	0	1	1	1
	Close	0	0	0	1	1	1

How did we get there?

We started with just an idea – we knew we wanted to make a quantum simulator, but that was about it.

From there, we mapped everything we thought we could need.

Then, we iterated, iterated, and iterated some more.

That's how we went from this...

		Structors							
		Single Qbit Calculator	Multi Qbit Calculator	Transmutor	Qbit	Reader	Register	GUI	
F u n c t i o n a l s	Toggle (T)	1	0	0	0	0	0	0	
	Hadamard (H)	1	0	0	0	0	0	0	
	Not (X)	1	0	0	0	0	0	0	
	Cnot (C)	0	1	0	0	0	0	0	
	Swap (S)	0	1	0	0	0	0	0	
	Equals (E)	0	0	1	0	0	0	0	
	Reset (R)	0	0	1	0	0	0	0	
	Read	0	0	0	0	1	0	0	
	Draw	0	0	0	0	0	0	1	
	Populate	0	0	0	0	0	0	1	
	Close	0	0	0	0	0	0	1	

...to this!

		Structors					
		Single Qbit Calculator	Multi Qbit Calculator	Transmutor	Reader	GUI	Alert
F u n c t i o n a l s	Toggle (T)	1	0	0	0	0	0
	Calculate (C)	1	1	0	0	0	0
	Transmute	0	0	1	0	0	0
	Read	0	0	0	1	0	0
	Draw	0	0	0	1	1	1
	Close	0	0	0	1	1	1

The next steps?

Of course, the Modularity Matrix was only one step in the development process.

Simply designing a program doesn't magically bring it to life, neither does only putting code in a file.

Once we had the code, it was time to ensure it worked.

Model Based Statistical Testing

Why did we choose JUMBL?

- Speed
- Scalability
- Ability to easily handle many states
- Tools to generate minimum coverage tests

Generating the TML

As I'm sure you all know, to use JUMBL we had to define all of our states.

Neither of us are professionals in the field of quantum computing, nor did we have the time to do the math by hand.

We used the Qiskit library from IBM for the computation and our own skills to automate the generation.

```
($ fill(1) $)  
model QubitCalculator
```

```
source [1000|0]  
  "Not"           [0100|0]  
  ($0.05$) "Cnot" [1000|0]  
  "Hadamard"     [RR00|0]  
  "Toggle"       [1000|1]  
  ($0.05$) "Switch" [1000|0]  
  "Reset"        [Exit]  
  "Compute"      [Exit]
```

```
[0100|0]  
  "Not"           [1000|0]  
  "Cnot"          [0001|0]  
  "Hadamard"     [RL00|0]
```

...

Generating the tests

Once we were through with the hard part of generating the TML, we needed to utilize that information to create the tests.

We opted to generate all of the minimum coverage tests, and 50 tests each of the random and weighted tests.

This left us with 292 tests, which we agreed was rather thorough.

Some example tests

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

Trajectory: 0
Model: QubitCalculator
Key:
Method: random

Events: 2
Including failure information.
#

```
=====
```

Step: 1, Trajectory: 0
[1000|0]."Toggle"
Step: 2, Trajectory: 0
[1000|1]."Reset"

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

Trajectory: 0
Model: QubitCalculator
Key:
Method: coverage

Events: 449
Including failure information.
#

```
=====
```

Step: 1, Trajectory: 0
[1000|0]."Hadamard"
Step: 2, Trajectory: 0
[RR00|0]."Toggle"
Step: 3, Trajectory: 0
[RR00|1]."Switch"
...

What we used

[1000|0]."Toggle"

[1000|1]."Reset"

[1000|0]."Hadamard"

[RR00|0]."Toggle"

[RR00|1]."Switch"

... 446 more ...

What did this yield?

With some hindrances, we were able to get our homemade version of the code working.

With that, we were able to replace the Qiskit library from our code completely. All calculations are our own.

Now, for a short demonstration...

Work Cited

1. https://miro.medium.com/max/1000/1*pjDx_psU07k-1xaU2Sp10Q.png
 2. <http://quantum-playground.appspot.com/images/screenshot2.png>
 3. <http://media5.datahacker.rs/2020/03/Picture36-1-1024x949.jpg>
 4. https://miro.medium.com/max/1200/0*iMwxZak6ytR571oM.png
 5. <https://science.sciencemag.org/content/318/5856/1567/F1.large.jpg>
 6. <https://qiskit.org/textbook/ch-states/images/bloch.png>
-