Defining and displaying state vectors

Qiskit uses the Python programming language, Qiskit specifically it may be helpful to very briefly discuss matrix and vector computations in Python. In Python, matrix and vector computations can be performed using the array class from the NumPy library (which includes many additional components for numerical computation).
Here is an example of a code cell that defines two vectors, ket0 and ket1 , corresponding to the qubit state vectors |0⟩ and |1⟩ and displays their average.

We can also use array to create matrices that represent operations.
Matrix multiplication (including matrix-vector multiplication as a special case) can be performed using the matmul function from NumPy.

```
[2]
from numpy import array

ket0 = array([1, 0])
ket1 = array([0, 1])

display(ket0 / 2 + ket1 / 2)
```

✓ 0.1s Python

```
array([0.5, 0.5])
```

```
[3]
M1 = array([[1, 1], [0, 0]])
M2 = array([[1, 1], [1, 0]])

M1 / 2 + M2 / 2
```

✓ 0.0s Python

```
array([[1. , 1. ],
       [0.5, 0. ]])
```

```
[4]
from numpy import matmul

display(matmul(M1, ket1))
display(matmul(M1, M2))
display(matmul(M2, M1))
```

✓ 0.0s Python

```
array([1, 0])

array([[2, 1],
       [0, 0]])

array([[1, 1],
       [1, 1]])
```

Qiskit's State vector class provides functionality for defining and manipulating quantum state vectors. The following code cell imports the State vector class and defines a few vectors using it. (Note that we need the sqrt function from the NumPy library to compute the square roots for the vector u .)

```
from qiskit.quantum_info import Statevector
from numpy import sqrt

u = Statevector([1 / sqrt(2), 1 / sqrt(2)])
v = Statevector([(1 + 2.0j) / 3, -2 / 3])
w = Statevector([1 / 3, 2 / 3])

print("State vectors u, v, and w have been defined.")
```

[5] ✓ 4.6s Python

```
State vectors u, v, and w have been defined.
```

The State vector class provides a draw method for displaying state vectors, including latex and text options for different visualizations, as this code cell demonstrates. The State vector class also includes the is valid method, which checks to see if a given vector is a valid quantum state vector (i.e., that it has Euclidean norm equal to 1):

```
display(u.draw("latex"))
display(v.draw("text"))
```

[6] ✓ 0.9s Python

$$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

```
[ 0.33333333+0.66666667j,-0.66666667+0.j        ]
```

```
display(u.is_valid())
display(w.is_valid())
```

[7] ✓ 0.0s Python

```
True

False
```

Measurements of quantum states can be simulated in Qiskit, using the measure method from the State vector class. Running the measure method simulates a standard basis measurement. It returns the result of that measurement, plus the new quantum state of our system after that measurement. Measurement outcomes are probabilistic, so the same method can return different results. Try running the cell a few times to see this. As an aside, State vector will throw an error if the measure method is applied to an invalid quantum state vector.

```
v = Statevector([(1 + 2.0j) / 3, -2 / 3])
v.draw("latex")
```

[23] ✓ 0.0s Python

$$(\frac{1}{3} + \frac{2i}{3})|0\rangle - \frac{2}{3}|1\rangle$$

```
v=Statevector([(1 + 2.0j) / 3, -2 / 3])
v.draw("latex")
v.measure()
```

[29] ✓ 0.0s Python

```
(np.str_('1'),
 Statevector([ 0.+0.j, -1.+0.j],
             dims=(2,)))
```

State vector also comes with a sample counts method that allows for the simulation of any number of measurements on the system. For example, the following cell shows the outcome of measuring the vector v 1000 times, which (with high probability) results in the outcome 0 approximately 5 out of every 9 times (or about 556 of the 1000 trials) and the outcome 1 approximately 4 out of every 9 times (or about 444 out of the 1000 trials). The cell also demonstrates the plot histogram function for visualizing the results. Running the cell multiple times and trying different numbers of samples in place of 1000may be helpful for developing some intuition for how the number of trials influences the estimated probabilities.

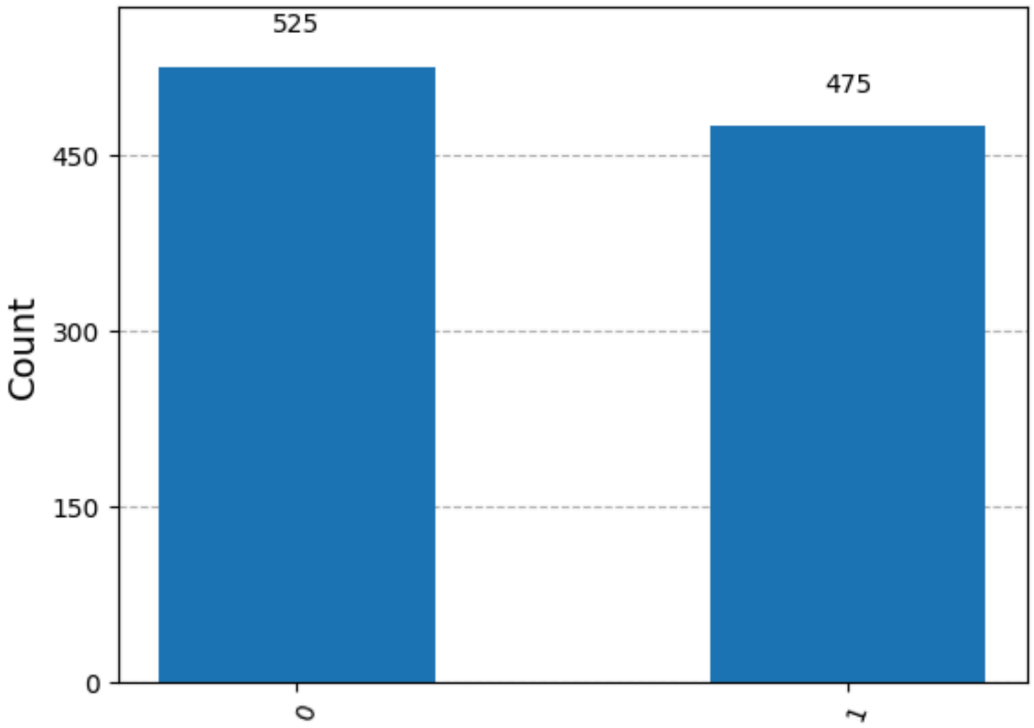
```
from qiskit.visualization import plot_histogram

statistics = v.sample_counts(1000)
display(statistics)
plot_histogram(statistics)
```

[10] ✓ 1.4s Python

```
{np.str_('0'): np.int64(525), np.str_('1'): np.int64(475)}
```

...


525
475
450
300
150
0
Count
0
1