Vectors Jupyter Notebook

Code Block 1 - For the first code block I am defining a vector with the use of numpy. This original code block can be found on page 2 of the python companion and the theory of this can be found on page 3 of VMLS. The original code block can be seen below.

In [21]:
```
import numpy as np
import random
x = [-1.1, 0.0, 3.6, -7.2]
len(x)
```

Out[21]:
```
4
```

For my example, I am generating a random integer value between 1 and 100 for each element of the vector and then printing the vector itself along with the length of the vector.

In [22]:
```
import numpy as np
import random
# Vector definition
cool_vector = np.array([random.randint(1, 100), random.randint(1, 100), random.
# Print statements
print("Random Cool Vector:", cool_vector)
print("Random Cool Vector Length:", len(cool_vector))
```

```
Random Cool Vector: [39 61 16]
Random Cool Vector Length: 3
```

Code Block 2 - For the second code block I am copying a vector with the use of the "copy()" method. Once the vector has been copied, we re-assign an element with indexing one of the elements and assign it to a new value. This original code block can be found on page 4 of the python companion. The theory of how to index a vector can be found on page 5 of VMLS. The original code block can be found below.

In [23]:
```
import numpy as np
x = np.array([-1.1, 0.0, 3.6, -7.2])
y = x.copy()
x[2] = 20.0
print(y)
```

```
[-1.1  0.   3.6 -7.2]
```

For my rendition of this concept, I first create a vector and then copy it to another vector. I then re-assign a random element from the copied vector with the use of "random" and then assign it a random integer that ranges between 7 and 10.

In [24]:
```
import numpy as np
import random
# Vector definition
another_cool_vector = np.array([1, 3, 5, 6])
# Copy the vector
copy_of_another_cool_vector = another_cool_vector.copy()
copy_of_another_cool_vector[random.randint(0, len(copy_of_another_cool_vector)
# Print the original and the copied vector that has been modified
```

```
print("Original Cool Vector:", another_cool_vector)
print("Copy Of Cool Vector:", copy_of_another_cool_vector)
```

```
Original Cool Vector: [1 3 5 6]
Copy Of Cool Vector: [ 1  3  5 10]
```

Code Block 3 - For the third code block, I am looking into concatenation of two vectors. The way that this is done in python can be found in python companion on page 6 and the theory of stacked vectors can be found on page 4 on VMLS. The original code block can be found below.

In [25]:
```
import numpy as np
x = np.array([1, -2])
y = np.array([1,1,0])
z = np.concatenate((x,y))
print(z)
```

```
[ 1 -2  1  1  0]
```

For my rendition of this, I first create two vectors. The first vector contains integer values from 1 to 4 and the second is an addition of four to each element of the original vector. I then concatenate the two vectors and print all of the vectors.

In [26]:
```
import numpy as np
# Vector definition
x = np.array([1, 2, 3, 4])
y = np.array([x[0] + 4, x[1] + 4, x[2] + 4, x[3] + 4])
# Concatenate vector
z = np.concatenate((x,y))
# Print the vectors
print("Vector X:", x)
print("Vector Y:", y)
print("Concatenated Vector, Z:", z)
```

```
Vector X: [1 2 3 4]
Vector Y: [5 6 7 8]
Concatenated Vector, Z: [1 2 3 4 5 6 7 8]
```

Code Block 4 - For the fourth code block we are examining the addition of two vectors. The original code block can be found in the python companion on page 10. The theory for this can be found on page 11 of VMLS. The original code block can be found below.

In [27]:
```
import numpy as np
x = np.array([1,2,3])
y = np.array([100,200,300])
print('Sum of arrays:', x+y)
print('Difference of arrays:', x-y)
```

```
Sum of arrays: [101 202 303]
Difference of arrays: [ -99 -198 -297]
```

My rendition of this example involves two vectors. The first vector is defined with set values, the second vector then has its elements assigned with random values that are predicated on the original vector. For example, for the first element of b, we take the first element of a as the minimum in the random calculation and then the maximum is calculated by taking a

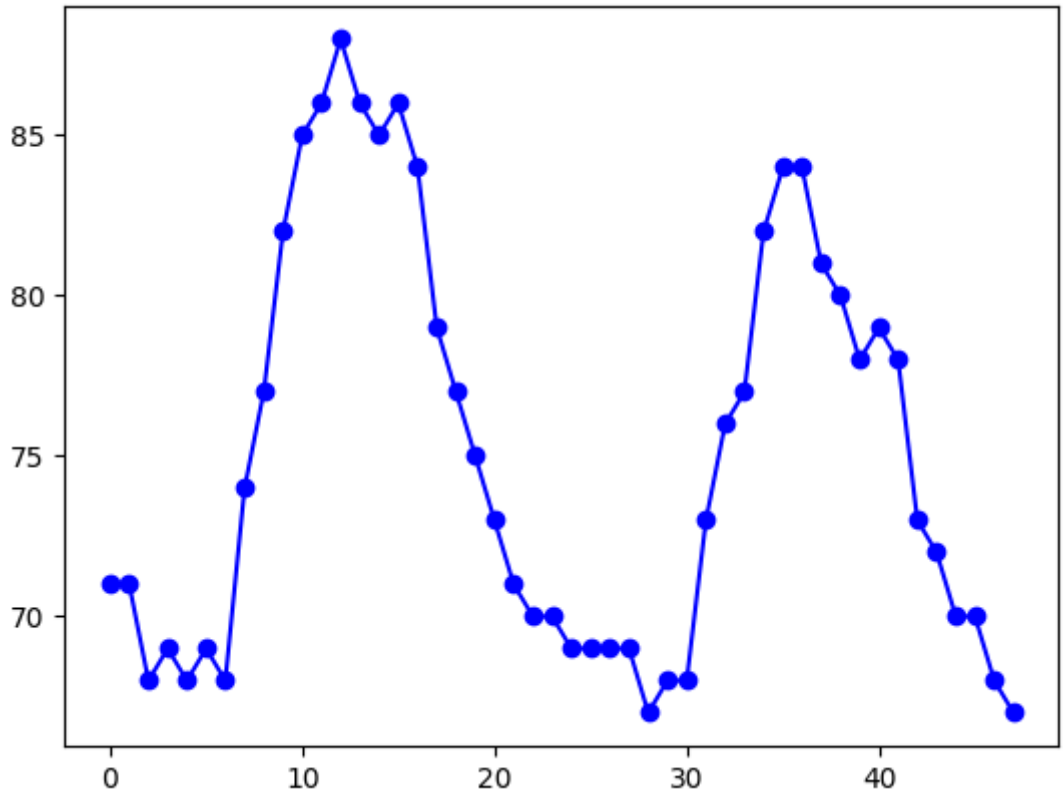
multiple of the second element from a and then multiplying it with the first element of a. This patten is repeated for all of the succeeding values.

In [28]:
```
import numpy as np
import random
# Vector definition
a = np.array([2, 4, 6, 8])
b = np.array([random.randint(a[0], a[1] * a[0]), random.randint(a[1], a[0] * a[
# Sum and difference
print("Sum of a and b:", a + b)
print("Difference of arrays:", a - b)
```

```
Sum of a and b: [ 9  8 15 22]
Difference of arrays: [-5  0 -3 -6]
```

Code Block 5 - For the fifth code block, I am taking the plot example from pages 9 and 10 of the python companion. The theory of this can be found on page 15 of VMLS. The original code block can be seen below.

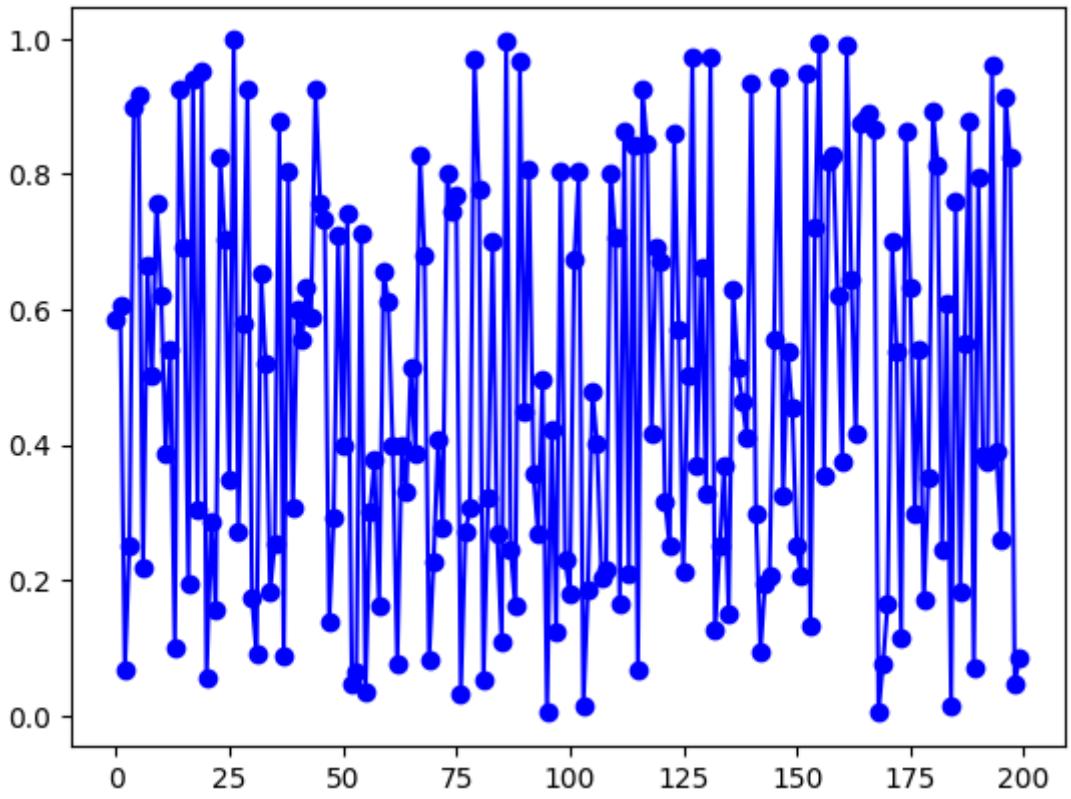
In [29]:
```
import matplotlib.pyplot as plt
plt.ion()
temps = [ 71, 71, 68, 69, 68, 69, 68, 74, 77, 82, 85, 86, 88, 86,
85, 86, 84, 79, 77, 75, 73, 71, 70, 70, 69, 69, 69, 69, 67,
68, 68, 73, 76, 77, 82, 84, 84, 81, 80, 78, 79, 78, 73, 72,
70, 70, 68, 67 ]
plt.plot(temps, '-bo')
plt.savefig('temperature.pdf', format = 'pdf')
plt.show()
```



Now, for my rendition I am going to plot the values of two vectors that have been added together. These vectors are 100 elements in length and the elements in them are random.

This code block uses the "append" method found later on the python companion.

In [30]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
# Vector definitions
a = []
b = []
# Assign random values to elements in vector
for i in range(100):
    a.append(random.random())
    b.append(random.random())
# Add vectors
c = a + b
# Plot
plt.plot(c, '-bo')
plt.show()
```



Code Block 6 - For this code block, I am going to take a look at scalar-vector multiplication. The original code block can be found on page 11 of the python companion. The theory of this operation can be found on page 15 of VMLS. The original code block can be found below.

In [31]:
```
import numpy as np
x = np.array([1,2,3])
print(2.2*x)
```

```
[2.2 4.4 6.6]
```

For my rendition, I am going to perform scalar-vector multiplication based off of the vector size.

In [32]:
```
import numpy as np
x = np.array([1, 2, 4, 5, 6, 7])
y = len(x) * x
print("Original Vector:", x)
print("Scalar Multiplied Vector:", y)
```

```
Original Vector: [1 2 4 5 6 7]
Scalar Multiplied Vector: [ 6 12 24 30 36 42]
```

Code Block 7 - For the seventh code block, I am going to dive into linear combination. The original code block can be found on page 12 of the python companion and the theory of this can be found on page 17 of VMLS. The original code block can be found below.

In [33]:
```
import numpy as np
a = np.array([1,2])
b = np.array([3,4])
alpha = -0.5
beta = 1.5
c = alpha*a + beta*b
print(c)
```

```
[4. 5.]
```

For my rendition, I am generating two random vectors that are comprised of 10 elements. These vectors are then scaled off of the middle element of the vector and then combined into a third vector.

In [34]:
```
import numpy as np
import random
# Vector definition
a = []
b = []
# Append random values
for i in range(10):
    a.append(random.random())
    b.append(random.random())
# Grab middle element
alpha = a[4]
beta = b[4]
# Create numpy arrays
a = np.array(a)
b = np.array(b)
# Calculate new vector
c = alpha*a + beta*b
print("Vector a:", a)
print("Vector b:", b)
print("Vector c:", c)
print("Alpha:", alpha)
print("Beta:", beta)
```

```
Vector a: [0.85226249 0.33766032 0.39193881 0.987842   0.84522741 0.07673927
 0.47095763 0.24721789 0.86380362 0.7840817 ]
Vector b: [0.22896021 0.65161428 0.09290719 0.6868429  0.51744293 0.52351399
 0.35781774 0.90120774 0.00718673 0.72531563]
Vector c: [0.83882945 0.62257295 0.37935159 1.19035314 0.98215656 0.33575074
 0.58321656 0.67527891 0.73382921 1.03803679]
Alpha: 0.8452274084651978
Beta: 0.5174429306003135
```

Code Block 8 - This code block is based off of the inner product example that can be found on page 14 of the python companion. The theory of this can be found on page 20 of VMLS. The original code block can be found below.

In [35]:
```
import numpy as np
c = np.array([0.1,0.1,0.1,1.1]) #cash flow vector
n = len(c)
r = 0.05 #5% per-period interest rate
d = np.array([(1+r)**-i for i in range(n)])
NPV = c @ d
print(NPV)
```

```
1.236162401468524
```

The only difference from the original code block to my rendition is that the interest rate is randomly generated. The minimum value of this random interest rate is 0.01 and the maximum is 0.10. The way that this number was generated can be found at the following link: https://www.w3schools.com/python/ref_random_uniform.asp

In [36]:
```
import numpy as np
import random
c = np.array([0.1,0.1,0.1,1.1]) #cash flow vector
n = len(c)
r = random.uniform(0.01, 0.1)
d = np.array([(1+r)**-i for i in range(n)])
NPV = c @ d
print(NPV)
```

```
1.3309583328199663
```

Code 9 - This code block examines slicing in python. The original code block can be found on page 7 of the python companion. The theory of slicing can be found on page 4 of VMLS. The original code block can be found below.

In [37]:
```
import numpy as np
x = np.array([1,8,3,2,1,9,7])
d = x[1:] - x[:-1]
print(d)
```

```
[ 7 -5 -1 -1  8 -2]
```

For my rendition, I am adding the values instead of subtracting them.

In [38]:
```
import numpy as np
x = np.array([1,8,3,2,1,9,7])
d = x[1:] + x[:-1]
print(d)
```

```
[ 9 11  5  3 10 16]
```

Code Block 10 - For the last code block, we are going to look at unit vectors. The application of this can be found on page 8 of the python companion. The theory of this can be found on page 5 of VMLS. The original code block can be found below.

In [39]:
```
import numpy as np
i = 2
n = 4
x = np.zeros(n)
x[i] = 1
print(x)
```

```
[0. 0. 1. 0.]
```

For my rendition, I am going to randomly create a unit vector. e.g. The length of the vector will be an arbitrary number of elements from 5 to 10 and location of i will be randomly selected based on the size of the vector.

In [40]:
```
import numpy as np
import random
n = random.randint(5, 10)
i = random.randint(0, n - 1)
x = np.zeros(n)
x[i] = 1
print(x)
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

Most of my examples involved random number generations. These random numbers followed the method that is found on page 9 of the python companion. Most of theory that was taken place can be found in the corresponding pages of VMLS.