Use the Mathematical Formula to Normalize a Vector in Python

In this method, we will compute the vector norm of an array using the mathematical formula. When we divide the array with this norm vector, we get the normalized vector. The following code implements this.

In [3]:
```
import numpy as np

v = np.array([3,4])

normalized_v = v / np.sqrt(np.sum(v**2))
print(normalized_v)
```

```
[0.6 0.8]
```

Use the numpy.linalg.norm() Function to Normalize a Vector in Python

The NumPy module in Python has the linalg.norm() function that can return the array's vector norm.Then we divide the array with this norm vector to get the normalized vector. For example, in the code below, we will create a random array and find its normalized form using this method.

In [4]:
```
import numpy as np

v = np.array([3,4])
normalized_v = v/np.linalg.norm(v)
print(normalized_v)
```

```
[0.6 0.8]
```

Use the sklearn.preprocessing.normalize() Function to Normalize a Vector in Python

The sklearn module has efficient methods available for data preprocessing and other machine learning tools. The normalize() function in this library is usually used with 2-D matrices and provides the option of L1 and L2 normalization.The code below will use thisfunction with a 1-D array and find its normalized form.

In [6]:
```
pip install sklearn
```

```
Collecting sklearn
   Downloading sklearn-0.0.postl.tar.gz (3.6 kB)
 Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py): started
  Building wheel for sklearn (setup.py): finished with status 'done'
  Created wheel for sklearn: filename=sklearn-0.0.postl-py3-none-any.whl size=2344 sha256=7072a653291935ec9f3feelef7f3247be1662
181a29f51196f3f259db667f217
  Stored in directory: c:\users\yue\appdata\local\pip\cache\wheels\f8\e0\3d\9d0c2020c44a519b9f02ab4fa6d2a4a996c98d79ab2f569fal
Successfully built sklearn
Installing collected packages: sklearn
Successfully installed sklearn-0.0.postl
Note: you may need to restart the kernel to use updated packages.
```

In [7]:
```
import numpy as np
from sklearn.preprocessing import normalize

v = np.array([3,4])
normalized_v = normalize(v[:,np.newaxis], axis=0).ravel()
print(normalized_v)
```

```
[0.6 0.8]
```

Inner Product

An inner product is a generalization of the dot product. In a vector space, it is a way to multiply vectors together, with the result of this multiplication being a scalar.

More precisely, for a real vector space, an inner product $\langle \cdot, \cdot \rangle$ satisfies the following four properties. Let u, v, and w be vectors and α be a scalar, then:

1. $\langle u+v, w \rangle = \langle u, w \rangle + \langle v, w \rangle$.
2. $\langle \alpha v, w \rangle = \alpha \langle v, w \rangle$.
3. $\langle v, w \rangle = \langle w, v \rangle$.
4. $\langle v, v \rangle \geq 0$ and equal if and only if $v = 0$.

In [16]:
```
# Example:
a=np.array([1,2,3])
b=np.array([0,1,0])
print(np.inner(a,b))
```

```
2
```

In [18]:
```

a= np.array([[1,2], [3,4]])

print ('Array a:')
print (a)
b = np.array([[11, 12], [13, 14]])

print ( 'Array b:' )
print (b)

print ('Inner product:'
print (np.inner(a,b))
print (np.inner(b,a))
```

```
Array a:
[[1  2]
 [3  4]]
Array b:
[[11 12]
 [13  14]]
Inner product:
[[35  41]
 [81  95]]
[ [35 81]
 [41  95]]
```

Outer Product

In [21]:
```
# Vectors
a= np.array([2, 6])
b = np.array([3, 10])
print( "Vectors :")
print("a =",a)
print("\nb = ", b)

# Outer product of vectors
print("\nOuter product of vectors a and b =")
print(np.outer(a, b))
```

```
Vectors  :
a     [2  6]

b     [ 3 10]

Outer product of vectors a and b
= [[ 6 20]
 [18  60]]
```

```
In [26]: # Matrices
         x = np.array([[3, 6, 4],[9, 4, 6]])
         y = np.array([[1, 15, 7],[3, 10, 8]])
         print("\nMatrices :")
         print("x =", x)
         print("\ny =", y)

         # Outer product of matrices
         print("\nOuter product of matrices x and y =")
         print(np.outer(x, y))
```

```
Matrices  :
x = [[3  6 4]
 [9  4  6]]

y = [[ 1 15  7]
 [ 3 10  8]]

Outer product of matrices x and y
[[   3  45  21   9  30  24]
 [   6  90  42  18  60  48]
 [   4  60  28  12  40  32]
 [  9 135  63  27  90  72]
 [   4  60  28  12  40  32]
 [   6  90  42  18  60  48]]
```

Cross Product

```
In [24]: # Vectors
         a= np.array([3, 6])
         b = np.array([9, 10])
         print( "Vectors :")
         print("a =",a)
         print("\nb =  ", b)

         #Crossproduct of vectors
         print("\nCross product of vectors a and b =")
         print(np.cross(a, b))
```

```
Vectors
a    [3  6]

b    [ 9 10]

Cross product of vectors a and b =
-24
```

```
In [25]: # Matrices
         x = np.array([[2, 6, 9],[2, 7, 3]])
         y = np.array([[7, 5, 6],[3, 12, 3]])
         print("\nMatrices :")
         print("x =", x)
         print("\ny =", y)

         #Crossproduct of matrices
         print("\nCross product of matrices x and y =")
         print(np.cross(x, y))
```

```
Matrices  :
x = [[2  6 9]
 [2  7  3]]

y = [[ 7   5   6]
 [ 3 12  3]]

Cross product of matrices x and y
[[ -9  51 -32]
 [-15   3   3]]
```

Angles and orthogonality

What are orthogonal vectors?

Two vectors u and v are considered to be **orthogonal** when the angle between them is $90°$. In other words, orthogonal vectors are perpendicular to each other. Orthogonality is denoted by $u \perp v$.

In [27]:
```
## Create two vectors

v1  np.array([1,-2, 4])
v2  np.array([2, 5, 2])
```

In [28]:
```
## Dot Product of V1 and V2
dot_product = np.sum(v1 * v2)

print("The dot product of v1 and v2 is", dot_product)
```

```
The dot product of v1 and v2 is 0
```

In [31]:
```
## Method 2 to calculate dot product of v1 and v2
dot_product1=np.dot(v1,v2)
print("The dot product of v1 and v2 is", dot_product1
```

```
The dot product of v1 and v2 is 0
```

Since the dot product is 0, the vectors are orthogonal.

In [32]:
```
#Let's also calculate the Length of each vector using the formula given above:

n1  np.sqrt(np.sum(v1 * v1))
n2  np.sqrt(np.sum(v2 * v2))

print('The L2 norm of v1 is', n1)
print('The L2 norm of v2 is', n2)
```

```
The L2 norm of v1 is 4.58257569495584
The L2 norm of v2 is 5.744562646538029
```

In [33]:
```
# Alternatively, the Length of a vector can be calculated using the L2 norm function builtin to Numpy:
n1  np.linalg.norm(v1, ord=2)
n2 = np.linalg.norm(v2, ord=2)

print('The L2 norm of v1 is', n1)
print('The L2 norm of v2 is', n2)
```

```
The L2 norm of v1 is 4.58257569495584
The L2 norm of v2 is 5.744562646538029
```

The dot product of the two vectors is 0, but the L2 norm of each vector is not equal to 1.This tells us that these vectors are orthogonal but not orthonormal.

Projection of a Vector on another vector

In [34]:
```
u   np.array([1, 2, 3])  # vector u
v   np.array([5, 6, 2])  # vector v:

# Task: Project vector u on vector v

# finding norm of the vector v
v_norm = np.sqrt(sum(v**2))

# Apply the formula as mentioned above
# for projecting a vector onto another vector
# find dot product using np.dot()
proj_of_u_on_v = (np.dot(u, v)/v_norm**2)*v

print("Projection of Vector u on Vector vis: "  proj_of_u_on_v)
```

```
Projection of Vector u on Vector vis:  [1.76923077 2.12307692 0.70769231]
```

Projection of a Vector onto a Plane

```
In [35]: # vector u
         u = np.array([2, 5, 8])

         # vector n: n is orthogonal vector to Plane P
         n = np.array([1, 1, 7])

         # Task: Project vector u on Plane P

         # finding norm of the vector n
         n_norm = np.sqrt(sum(n**2))

         # Apply the formula as mentioned above
         # for projecting a vector onto the orthogonal vector n
         # find dot product using np.dot()
         proj_of_u_on_n = (np.dot(u, n)/n_norm**2)*n

         # subtract proj_of_u_on_n from u:
         # this is the projection of u on Plane P
         print(""Projection of Vector u on Plane P is:u - proj_of_u_on_n)

         Projection of Vector u on Plane Pis: 0.76470588  3.76470588 -0.64705882]

In [ ]:
```

Generalization: complement an m-basis in a n-D space

In an n-dimensional space. given an (n, m) orthonormal basis x with m s.t. 1 <= m < n (in other words, m vectors in a n-dimensional space put together as columns of x): find n - m vectors that are orthonormal, and that are all orthogonal to x. We can do this in one shot using SVD.

```
In [1]: import numpy as np
        # 1.  setup
        #    we use SVD for the setup as well, for convenience,
        #    but it's not necessary at all. It is sufficient that
        #    x, T@  x == I

        n, m = 6, 2 # for example
        x,  _,  _ = np.linalg.svd(np.random.uniform(size=(n, m)))
        x = x[:,  :m]



In
[2]:
        # 2. check
        >>> np.allclose(x.T@ x, np.eye(m))
        True

Out[2]: True


In [4]: »> x.shape
        (6,  2)
        # So, at this point, xis orthonormal and of shape (n, m).

Out[4]: (6,  2)
```

Find y to be one (of possibly many) orthonormal basis that is orthogonal to x:

```
In [12]: # 3. solve
         u, s, v = np.linalg.svd(x)
         y = u[:, m:]

         a=np.dot(np.dot(y.T,y))

         # 4. check
         >>> np.allclose((np.dot(y.T,y)), np.eye(n-m))
         True

         >>> np.allclose((np.dot(x.T,y)), 0)
         True

           File "<ipython-input-12-349f1f232b94>", line 6
             >>> np.allclose((np.dot(y.T,y)), np.eye(n-m))

         SyntaxError: invalid syntax
```

In [ ]:
```
# 3. solve
u, s, v = np.linalg.svd(x)
y = u[:, m:]

# 4. check
>>> np.allclose(y.T@ y, np.eye(n-m))
True

>>> np.allclose(x.T@ y, 0)
True
```

In [