PY0101EN-5-1-Numpy1D

May 12, 2022

1 1D Numpy in Python

Estimated time needed: **30** minutes

1.1 Objectives

After completing this lab you will be able to:

- Import and use the `numpy` library
- Perform operations with `numpy`

Table of Contents

```
<ul>
    <li><a href="https://pre/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=0000
    <li>
        <a href="https://numpy/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000
        <ul>
            <li><a href="https://type/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_cont
            <li><a href="https://val/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_conte
            <li><a href="https://slice/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_co
            <li><a href="https://list/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_cont
            <li><a href="https://other/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_co
        </ul>
    </li>
    <li>
        <a href="op">Numpy Array Operations</a>
        <ul>
            <li><a href="https://add/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_conte
            <li><a href="https://multi/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_co
            <li><a href="https://prod/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_cont
            <li><a href="https://dot/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_conte
            <li><a href="https://cons/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_cont
        </ul>
    </li>
    <li><a href="https://math/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000
    <li><a href="https://lin/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=0000
</ul>
```

1.1.1 Create a Python List as follows:

[1]:
```
# Create a python list

a = ["0", 1, "two", "3", 4]
```

We can access the data via an index:

We can access each element using a square bracket as follows:

[2]:
```
# Print each element

print("a[0]:", a[0])
print("a[1]:", a[1])
print("a[2]:", a[2])
print("a[3]:", a[3])
print("a[4]:", a[4])
```

```
a[0]: 0
a[1]: 1
a[2]: two
a[3]: 3
a[4]: 4
```

What is Numpy?

NumPy is a Python library used for working with arrays, linear algebra, fourier transform, and matrices.A numpy array is similar to a list. NumPy stands for Numerical Python and it is an open source project.The array object in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

NumPy is usually imported under the np alias.

It's usually fixed in size and each element is of the same type. We can cast a list to a numpy array by first importing `numpy`:

[3]:
```
# import numpy library

import numpy as np
```

We then cast the list as follows:

[4]:
```
# Create a numpy array

a = np.array([0, 1, 2, 3, 4])
a
```

[4]:
```
array([0, 1, 2, 3, 4])
```

Each element is of the same type, in this case integers:

As with lists, we can access each element via a square bracket:

[5]:
```
# Print each element

print("a[0]:", a[0])
print("a[1]:", a[1])
print("a[2]:", a[2])
print("a[3]:", a[3])
print("a[4]:", a[4])
```

```
a[0]: 0
a[1]: 1
a[2]: 2
a[3]: 3
a[4]: 4
```

1.1.2 Checking NumPy Version

The version string is stored under **version** attribute.

[6]:
```
print(np.__version__)
```

```
1.21.6
```

Type

If we check the type of the array we get numpy.ndarray:

[7]:
```
# Check the type of the array

type(a)
```

[7]:
```
numpy.ndarray
```

As numpy arrays contain data of the same type, we can use the attribute "dtype" to obtain the data type of the array's elements. In this case, it's a 64-bit integer:

[8]:
```
# Check the type of the values stored in numpy array

a.dtype
```

[8]:
```
dtype('int64')
```

1.1.3 Try it yourself

Check the type of the array and Value type for the given array **c**

[84]:
```
b = np.array([3.1, 11.02, 6.2, 213.2, 5.2])

# Enter your code here
```

```
type(b)
```

[84]:
```
numpy.ndarray
```

Click here for the solution

```
type(b)
```

```
b.dtype
```

```
If we examine the attribute dtype we see float 64, as the elements are not integers:
```

Assign value

We can change the value of the array. Consider the array c:

[10]:
```
# Create numpy array

c = np.array([20, 1, 2, 3, 4])
c
```

[10]:
```
array([20,  1,  2,  3,  4])
```

We can change the first element of the array to 100 as follows:

[11]:
```
# Assign the first element to 100

c[0] = 100
c
```

[11]:
```
array([100,   1,   2,   3,   4])
```

We can change the 5th element of the array to 0 as follows:

[12]:
```
# Assign the 5th element to 0

c[4] = 0
c
```

[12]:
```
array([100,   1,   2,   3,   0])
```

1.1.4 Try it yourself

Assign the value 20 for the second element in the given array.

[85]:
```
a = np.array([10, 2, 30, 40,50])

# Enter your code here
a[1] = 20
```

Click here for the solution

```
a[1]=20
a
```

Slicing

Like lists, we can slice the numpy array. Slicing in python means taking the elements from the given index to another given index.

We pass slice like this: [start:end].The element at end index is not being included in the output.

We can select the elements from 1 to 3 and assign it to a new numpy array d as follows:

[14]:
```
# Slicing the numpy array

d = c[1:4]
d
```

[14]:
```
array([1, 2, 3])
```

We can assign the corresponding indexes to new values as follows:

[15]:
```
# Set the fourth element and fifth element to 300 and 400

c[3:5] = 300, 400
c
```

[15]:
```
array([100,   1,   2, 300, 400])
```

We can also define the steps in slicing, like this: [start:end:step].

[16]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5:2])
```

```
[2 4]
```

If we don't pass start its considered 0

[17]:
```
print(arr[:4])
```

```
[1 2 3 4]
```

If we don't pass end it considers till the length of array.

[18]:
```
print(arr[4:])
```

```
[5 6 7]
```

If we don't pass step its considered 1

[19]:
```
print(arr[1:5:])
```

```
[2 3 4 5]
```

1.1.5 Try it yourself

Print the even elements in the given array.

[20]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

# Enter your code here
```

Click here for the solution

```
print(arr[1:8:2])
```

Assign Value with List

Similarly, we can use a list to select more than one specific index. The list `select` contains several values:

[21]:
```
# Create the index list

select = [0, 2, 3, 4]
select
```

[21]:
```
[0, 2, 3, 4]
```

We can use the list as an argument in the brackets. The output is the elements corresponding to the particular indexes:

[22]:
```
# Use List to select elements

d = c[select]
d
```

[22]:
```
array([100,   2, 300, 400])
```

We can assign the specified elements to a new value. For example, we can assign the values to 100 000 as follows:

[23]:
```
# Assign the specified elements to new value

c[select] = 100000
c
```

[23]:
```
array([100000,      1, 100000, 100000, 100000])
```

Other Attributes

Let's review some basic array attributes using the array a:

[24]:
```
# Create a numpy array

a = np.array([0, 1, 2, 3, 4])
a
```

[24]:
```
array([0, 1, 2, 3, 4])
```

The attribute size is the number of elements in the array:

[25]:
```
# Get the size of numpy array

a.size
```

[25]:
```
5
```

The next two attributes will make more sense when we get to higher dimensions but let's review them. The attribute ndim represents the number of array dimensions, or the rank of the array. In this case, one:

[26]:
```
# Get the number of dimensions of numpy array

a.ndim
```

[26]:
```
1
```

The attribute shape is a tuple of integers indicating the size of the array in each dimension:

[27]:
```
# Get the shape/size of numpy array

a.shape
```

[27]:
```
(5,)
```

1.1.6 Try it yourself

Find the size ,dimension and shape for the given array **b**

[28]:
```
b = np.array([10, 20, 30, 40, 50, 60, 70])

# Enter your code here
```

Click here for the solution

```
b.size

b.ndim

b.shape
```

1.1.7 Numpy Statistical Functions

[29]:
```
# Create a numpy array

a = np.array([1, -1, 1, -1])
```

[30]:
```
# Get the mean of numpy array

mean = a.mean()
mean
```

[30]: 0.0

[31]:
```
# Get the standard deviation of numpy array

standard_deviation=a.std()
standard_deviation
```

[31]: 1.0

[32]:
```
# Create a numpy array

b = np.array([-1, 2, 3, 4, 5])
b
```

[32]: array([-1, 2, 3, 4, 5])

[33]:
```
# Get the biggest value in the numpy array

max_b = b.max()
max_b
```

[33]: 5

[34]:
```
# Get the smallest value in the numpy array

min_b = b.min()
min_b
```

[34]: -1

1.1.8 Try it yourself

Find the sum of maximum and minimum value in the given numpy array

[35]:
```
c = np.array([-10, 201, 43, 94, 502])

# Enter your code here
```

Click here for the solution

```
max_c = c.max()
max_c

min_c = c.min()
min_c


Sum = (max_c +min_c)
Sum
```

Numpy Array Operations

You could use arithmetic operators directly between NumPy arrays

Array Addition

Consider the numpy array u:

[36]:
```
u = np.array([1, 0])
u
```

[36]:
```
array([1, 0])
```

Consider the numpy array v:

[37]:
```
v = np.array([0, 1])
v
```

[37]:
```
array([0, 1])
```

We can add the two arrays and assign it to z:

[38]:
```
# Numpy Array Addition

z = np.add(u, v)
z
```

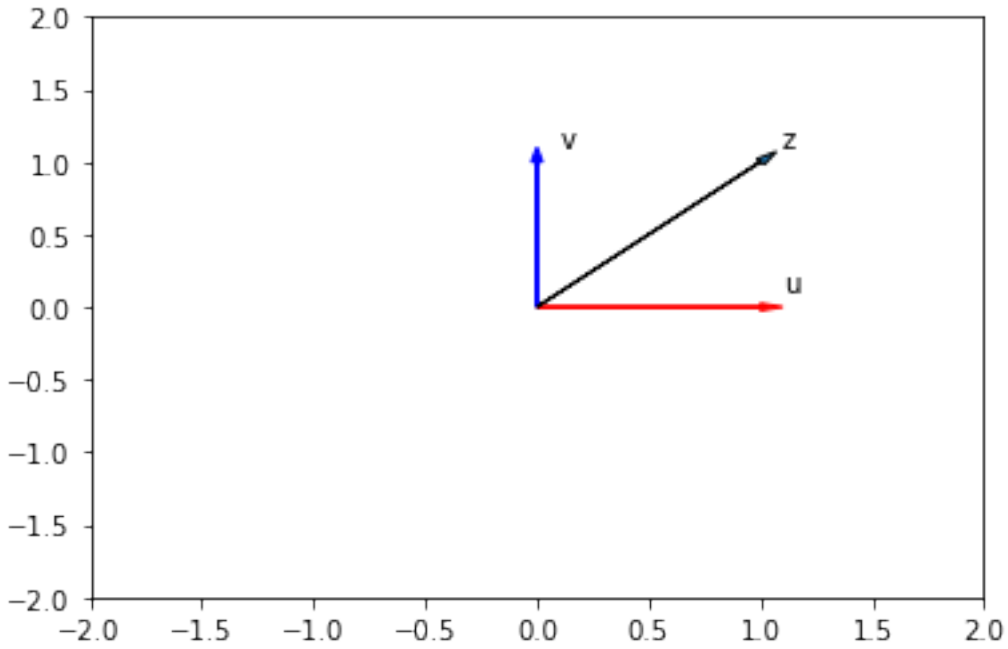
[38]:
```
array([1, 1])
```

The operation is equivalent to vector addition:

[39]:
```
# Plotting functions
```

```
import time
import sys
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

def Plotvec1(u, z, v):

    ax = plt.axes() # to generate the full window axes
    ax.arrow(0, 0, *u, head_width=0.05, color='r', head_length=0.1)# Add an␣
 ↪arrow to the  U Axes with arrow head width 0.05, color red and arrow head␣
 ↪length 0.1
    plt.text(*(u + 0.1), 'u')#Adds the text u to the Axes

    ax.arrow(0, 0, *v, head_width=0.05, color='b', head_length=0.1)# Add an␣
 ↪arrow to the  v Axes with arrow head width 0.05, color red and arrow head␣
 ↪length 0.1
    plt.text(*(v + 0.1), 'v')#Adds the text v to the Axes

    ax.arrow(0, 0, *z, head_width=0.05, head_length=0.1)
    plt.text(*(z + 0.1), 'z')#Adds the text z to the Axes
    plt.ylim(-2, 2)#set the ylim to bottom(-2), top(2)
    plt.xlim(-2, 2)#set the xlim to left(-2), right(2)
```

[40]:
```
# Plot numpy arrays

Plotvec1(u, z, v)
```



1.1.9 Try it yourself

Perform addition operation on the given numpy array arr1 and arr2:

[41]:
```
arr1 = np.array([10, 11, 12, 13, 14, 15])
arr2 = np.array([20, 21, 22, 23, 24, 25])

# Enter your code here
```

Click here for the solution

```
arr3 = np.add(arr1, arr2)
arr3
```

1.1.10 Array Subtraction

Consider the numpy array a:

[42]:
```
a = np.array([10, 20, 30])
a
```

[42]:
```
array([10, 20, 30])
```

Consider the numpy array b:

[43]:
```
b = np.array([5, 10, 15])
b
```

[43]:
```
array([ 5, 10, 15])
```

We can subtract the two arrays and assign it to c:

[44]:
```
c = np.subtract(a, b)

print(c)
```

```
[ 5 10 15]
```

1.1.11 Try it yourself

Perform subtraction operation on the given numpy array arr1 and arr2:

[90]:
```
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])

# Enter your code here
arr = arr2 - arr1
arr
```

[90]:
```
array([ 10,   1,  -8, -17, -26, -35])
```

Click here for the solution

```
arr3 = np.subtract(arr1, arr2)
arr3
```

Array Multiplication

Consider the vector numpy array y:

[46]:
```
# Create a numpy array

x = np.array([1, 2])
x
```

[46]:
```
array([1, 2])
```

[47]:
```
# Create a numpy array

y = np.array([2, 1])
y
```

[47]:
```
array([2, 1])
```

We can multiply every element in the array by 2:

[48]:
```
# Numpy Array Multiplication

z = np.multiply(x, y)
z
```

[48]:
```
array([2, 2])
```

This is equivalent to multiplying a vector by a scaler:

1.1.12 Try it yourself

Perform multiply operation on the given numpy array arr1 and arr2:

[49]:
```
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([2, 1, 2, 3, 4, 5])

# Enter your code here
```

Click here for the solution

```
arr3 = np.multiply(arr1, arr2)
arr3
```

1.1.13 Array Division

Consider the vector numpy array a:

[50]:
```
a = np.array([10, 20, 30])
a
```

[50]:
```
array([10, 20, 30])
```

Consider the vector numpy array b:

[51]:
```
b = np.array([2, 10, 5])
b
```

[51]:
```
array([ 2, 10,  5])
```

We can divide the two arrays and assign it to c:

[52]:
```
c = np.divide(a, b)
c
```

[52]:
```
array([5., 2., 6.])
```

1.1.14 Try it yourself

Perform division operation on the given numpy array arr1 and arr2:

[53]:
```
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 5, 10, 8, 2, 33])

# Enter your code here
```

Click here for the solution

```
arr3 = np.divide(arr1, arr2)
arr3
```

Dot Product

The dot product of the two numpy arrays u and v is given by:

[54]:
```
X = np.array([1, 2])
Y = np.array([3, 2])
```

[55]:
```
# Calculate the dot product

np.dot(X, Y)
```

[55]: 7

[56]:
```
#Elements of X
print(X[0])
print(X[1])
```

```
1
2
```

[57]:
```
#Elements of Y
print(Y[0])
print(Y[1])
```

```
3
2
```

We are performing the dot product which is shown as below

1.1.15 Try it yourself

Perform dot operation on the given numpy array ar1 and ar2:

[58]:
```
arr1 = np.array([3, 5])
arr2 = np.array([2, 4])
```

```
# Enter your code here
```

Click here for the solution

```
arr3 = np.dot(arr1, arr2)
arr3
```

Adding Constant to a Numpy Array

Consider the following array:

[59]:
```
# Create a constant to numpy array

u = np.array([1, 2, 3, -1])
u
```

[59]:
```
array([ 1,  2,  3, -1])
```

Adding the constant 1 to each element in the array:

[60]:
```
# Add the constant to array

u + 1
```

[60]:
```
array([2, 3, 4, 0])
```

The process is summarised in the following animation:

1.1.16 Try it yourself

Add Constant 5 to the given numpy array ar:

[61]:
```
arr = np.array([1, 2, 3, -1])

# Enter your code here
```

Click here for the solution

```
arr + 5
```

Mathematical Functions

We can access the value of pi in numpy as follows :

[62]:
```
# The value of pi

np.pi
```

[62]:
```
3.141592653589793
```

We can create the following numpy array in Radians:

[63]:
```
# Create the numpy array in radians

x = np.array([0, np.pi/2 , np.pi])
```

We can apply the function sin to the array x and assign the values to the array y; this applies the sine function to each element in the array:

[64]:
```
# Calculate the sin of each elements

y = np.sin(x)
y
```

[64]:
```
array([0.0000000e+00, 1.0000000e+00, 1.2246468e-16])
```

Linspace

A useful function for plotting mathematical functions is linspace. Linspace returns evenly spaced numbers over a specified interval.

numpy.linspace(start, stop, num = int value)

start : start of interval range

stop : end of interval range

num : Number of samples to generate.

[65]:
```
# Makeup a numpy array within [-2, 2] and 5 elements

np.linspace(-2, 2, num=5)
```

[65]:
```
array([-2., -1.,  0.,  1.,  2.])
```

If we change the parameter num to 9, we get 9 evenly spaced numbers over the interval from -2 to 2:

[66]:
```
# Make a numpy array within [-2, 2] and 9 elements

np.linspace(-2, 2, num=9)
```

[66]:
```
array([-2. , -1.5, -1. , -0.5,  0. ,  0.5,  1. ,  1.5,  2. ])
```

We can use the function linspace to generate 100 evenly spaced samples from the interval 0 to 2 :

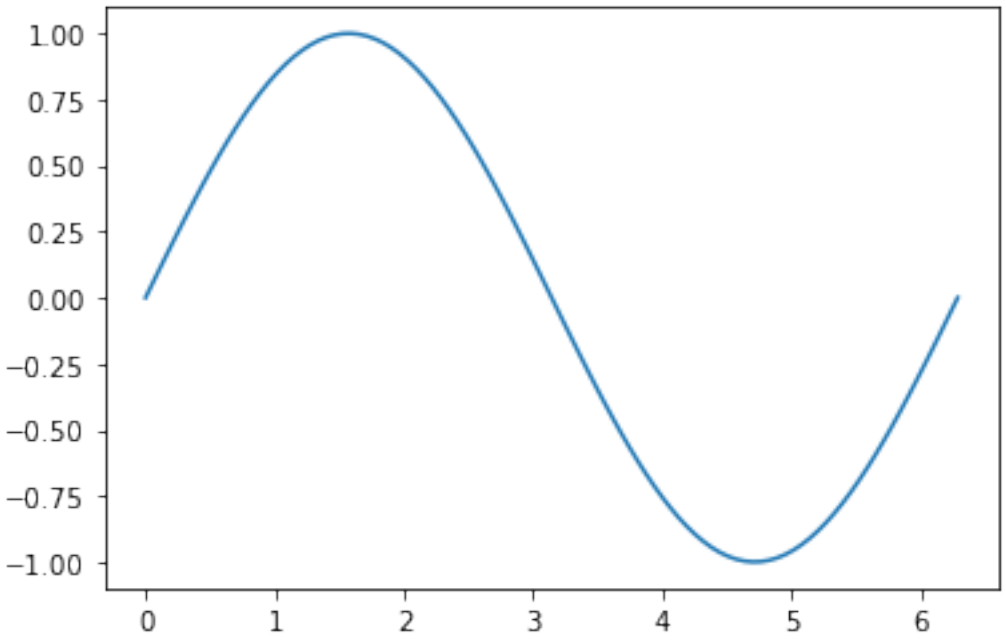
[67]:
```
# Make a numpy array within [0, 2 ] and 100 elements

x = np.linspace(0, 2*np.pi, num=100)
```

We can apply the sine function to each element in the array x and assign it to the array y:

[68]:
```
# Calculate the sine of x list

y = np.sin(x)
```

[69]:
```
# Plot the result

plt.plot(x, y)
```

[69]:
```
[<matplotlib.lines.Line2D at 0x7f0f93c0e3d0>]
```



1.1.17 Try it yourself

Make a numpy array within [5, 4] and 6 elements

[92]:
```
# Enter your code here
r = np.linspace(5,4,num=6)
```

Click here for the solution

```
np.linspace(5, 4, num=6)
```

1.1.18 Iterating 1-D Arrays

Iterating means going through elements one by one.

If we iterate on a 1-D array it will go through each element one by one.

If we execute the numpy array, we get in the array format

[71]:
```
arr1 = np.array([1, 2, 3])
print(arr1)
```

```
[1 2 3]
```

But if you want to result in the form of the list, then you can use for loop:

[72]:
```
for x in arr1:
  print(x)
```

```
1
2
3
```

Quiz on 1D Numpy Array

Implement the following vector subtraction in numpy: u-v

[73]:
```
# Write your code below and press Shift+Enter to execute

u = np.array([1, 0])
v = np.array([0, 1])
```

Click here for the solution

```
u - v
```

Multiply the numpy array z with -2:

[74]:
```
# Write your code below and press Shift+Enter to execute

z = np.array([2, 4])
```

Click here for the solution

```
-2 * z
```

Consider the list [1, 2, 3, 4, 5] and [1, 0, 1, 0, 1]. Cast both lists to a numpy array then multiply them together:

[75]:
```
# Write your code below and press Shift+Enter to execute
```

Click here for the solution

```
a = np.array([1, 2, 3, 4, 5])
b = np.array([1, 0, 1, 0, 1])
a * b
```

[76]:
```
# Import the libraries

import time
import sys
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

def Plotvec2(a,b):
    ax = plt.axes()# to generate the full window axes
    ax.arrow(0, 0, *a, head_width=0.05, color ='r', head_length=0.1)#Add an␣
 ↪arrow to the  a Axes with arrow head width 0.05, color red and arrow head␣
 ↪length 0.1
    plt.text(*(a + 0.1), 'a')
    ax.arrow(0, 0, *b, head_width=0.05, color ='b', head_length=0.1)#Add an␣
 ↪arrow to the  b Axes with arrow head width 0.05, color blue and arrow head␣
 ↪length 0.1
    plt.text(*(b + 0.1), 'b')
    plt.ylim(-2, 2)#set the ylim to bottom(-2), top(2)
    plt.xlim(-2, 2)#set the xlim to left(-2), right(2)
```

Convert the list [-1, 1] and [1, 1] to numpy arrays a and b. Then, plot the arrays as vectors using the fuction Plotvec2 and find their dot product:

[77]:
```
# Write your code below and press Shift+Enter to execute
```

Click here for the solution

```
a = np.array([-1, 1])
b = np.array([1, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a,b))
```

Convert the list [1, 0] and [0, 1] to numpy arrays a and b. Then, plot the arrays as vectors using the function Plotvec2 and find their dot product:

[78]:
```
# Write your code below and press Shift+Enter to execute
```

Click here for the solution

```
a = np.array([1, 0])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a, b))
```

Convert the list [1, 1] and [0, 1] to numpy arrays a and b. Then plot the arrays as vectors using the fuction Plotvec2 and find their dot product:

[79]:
```
# Write your code below and press Shift+Enter to execute
```

Click here for the solution

```
a = np.array([1, 1])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a, b))
```

Why are the results of the dot product for [-1, 1] and [1, 1] and the dot product for [1, 0] and [0, 1] zero, but not zero for the dot product for [1, 1] and [0, 1]?

Hint: Study the corresponding figures, pay attention to the direction the arrows are pointing to.

[80]:
```
# Write your code below and press Shift+Enter to execute
```

Click here for the solution

```
The vectors used for question 4 and 5 are perpendicular. As a result, the dot product is zero.
```

Convert the list [1, 2, 3] and [8, 9, 10] to numpy arrays arr1 and arr2. Then perform Addition , Subtraction , Multiplication , Division and Dot Operation on the arr1 and arr2.

[81]:
```
# Write your code below and press Shift+Enter to execute
```

Click here for the solution

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([8, 9, 10])

arr3 = np.add(arr1, arr2)
arr3

arr4 = np.subtract(arr1, arr2)
arr4

arr5 = np.multiply(arr1, arr2)
arr5


arr6 = np.divide(arr1, arr2)
arr6

arr7 = np.dot(arr1, arr2)
arr7
```

Convert the list [1, 2, 3, 4, 5] and [6, 7, 8, 9, 10] to numpy arrays arr1 and arr2. Then find the even and odd numbers from arr1 and arr2.

[82]:
```
# Write your code below and press Shift+Enter to execute
```

Click here for the solution

```
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([6, 7, 8, 9, 10])

#Starting index in slice is 1 as first even element(2) in array1 is at index 1
even_arr1 = arr1[1:5:2]
print("even for array1",even_arr1)

#Starting index in slice is 0 as first odd element(1) in array1 is at index 0
odd_arr1=arr1[0:5:2]
print("odd for array1",odd_arr1)

#Starting index in slice is 0 as first even element(6) in array2 is at index 0
even_arr2 = arr2[0:5:2]
print("even for array2",even_arr2)


#Starting index in slice is 1 as first odd element(7) in array2 is at index 1
odd_arr2=arr2[1:5:2]
print("odd for array2",odd_arr2)
```

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python.

1.2 Author

Joseph Santarcangelo

1.3 Other contributors

Mavis Zhou

1.4 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-03-08	2.2	Niveditha	Modified and added practice problem
2022-01-10	2.1	Malika	Removed the readme for GitShare
2020-08-26	2.0	Lavanya	Moved lab to course repo in GitLab

##

[ ]: