1D Numpy in Python

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

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

Table of Contents

Preparation

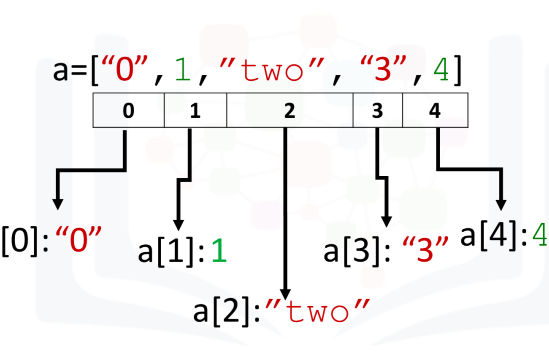
In [1]:
```
# Import the libraries

import time
import sys
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# Plotting functions

def Plotvec1(u, z, v):

    ax = plt.axes()
    ax.arrow(0, 0, *u, head_width=0.05, color='r', head_length=0.1)
    plt.text(*(u + 0.1), 'u')

    ax.arrow(0, 0, *v, head_width=0.05, color='b', head_length=0.1)
    plt.text(*(v + 0.1), 'v')
    ax.arrow(0, 0, *z, head_width=0.05, head_length=0.1)
    plt.text(*(z + 0.1), 'z')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)

def Plotvec2(a,b):
    ax = plt.axes()
    ax.arrow(0, 0, *a, head_width=0.05, color ='r', 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)
    plt.text(*(b + 0.1), 'b')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)
```

Create a Python List as follows:

In [3]:
```
# 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:

In [4]:
```
# 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?

A numpy array is similar to a list. 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:

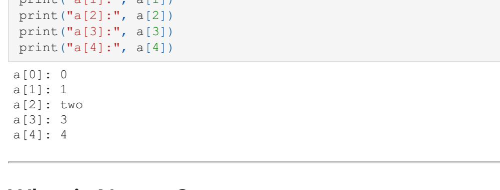
In [5]:
```
# import numpy library

import numpy as np
```

We then cast the list as follows:

In [6]:
```
# Create a numpy array

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

Out[6]: `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:

In [7]:
```
# 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
```

Type

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

In [8]:
```
# Check the type of the array

type(a)
```

Out[8]: `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 a 64-bit integer:

In [9]:
```
# Check the type of the values stored in numpy array

a.dtype
```

Out[9]: `dtype('int32')`

We can create a numpy array with real numbers:

In [10]:
```
# Create a numpy array

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

When we check the type of the array we get **numpy.ndarray**:

In [11]:
```
# Check the type of array

type(b)
```

Out[11]: `numpy.ndarray`

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

In [12]:
```
# Check the value type

b.dtype
```

Out[12]: `dtype('float64')`

Assign value

We can change the value of the array, consider the array `c`:

In [13]:
```
# Create numpy array

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

Out[13]: `array([20,  1,  2,  3,  4])`

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

In [14]:
```
# Assign the first element to 100

c[0] = 100
c
```

Out[14]: `array([100,   1,   2,   3,   4])`

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

In [15]:
```
# Assign the 5th element to 0

c[4] = 0
c
```

Out[15]: `array([100,   1,   2,   3,   0])`

Slicing

Like lists, we can slice the numpy array, and we can select the elements from 1 to 3 and assign it to a new numpy array `d` as follows:

In [16]:
```
# Slicing the numpy array

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

Out[16]: `array([1, 2, 3])`

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

In [17]:
```
# Set the fourth element and fifth element to 300 and 400

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

Out[17]: `array([100,   1,   2, 300, 400])`

Assign Value with List

Similarly, we can use a list to select a specific index. The list ' select ' contains several values:

In [18]:
```
# Create the index list

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

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

In [19]:
```
# Use List to select elements

d = c[select]
d
```

Out[19]: `array([100,   2, 300])`

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

In [20]:
```
# Assign the specified elements to new value

c[select] = 100000
c
```

Out[20]: `array([100000,      1, 100000, 100000,    400])`

Other Attributes

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

In [21]:
```
# Create a numpy array

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

Out[21]: `array([0, 1, 2, 3, 4])`

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

In [22]:
```
# Get the size of numpy array

a.size
```

Out[22]: `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:

In [23]:
```
# Get the number of dimensions of numpy array

a.ndim
```

Out[23]: `1`

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

In [24]:
```
# Get the shape/size of numpy array

a.shape
```

Out[24]: `(5,)`

In [25]:
```
# Create a numpy array

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

In [26]:
```
# Get the mean of numpy array

mean = a.mean()
mean
```

Out[26]: `0.0`

In [27]:
```
# Get the standard deviation of numpy array

standard_deviation=a.std()
standard_deviation
```

Out[27]: `1.0`

In [28]:
```
# Create a numpy array

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

Out[28]: `array([-1,  2,  3,  4,  5])`

In [29]:
```
# Get the biggest value in the numpy array

max_b = b.max()
max_b
```

Out[29]: `5`

In [30]:
```
# Get the smallest value in the numpy array

min_b = b.min()
min_b
```

Out[30]: `-1`

Numpy Array Operations

Array Addition

Consider the numpy array `u`:

In [31]:
```
u = np.array([1, 0])
u
```

Out[31]: `array([1, 0])`

Consider the numpy array `v`:

In [32]:
```
v = np.array([0, 1])
v
```

Out[32]: `array([0, 1])`

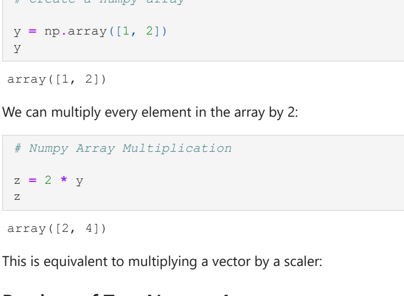
We can add the two arrays and assign it to z:

In [33]:
```
# Numpy Array Addition

z = u + v
z
```

Out[33]: `array([1, 1])`

The operation is equivalent to vector addition:

In [34]:
```
# Plot numpy arrays

Plotvec1(u, z, v)
```



Array Multiplication

Consider the vector numpy array `y`:

In [35]:
```
# Create a numpy array

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

Out[35]: `array([1, 2])`

We can multiply every element in the array by 2:

In [36]:
```
# Numpy Array Multiplication

z = 2 * y
z
```

Out[36]: `array([2, 4])`

This is equivalent to multiplying a vector by a scaler:

Product of Two Numpy Arrays

Consider the following array `u`:

In [37]:
```
# Create a numpy array

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

Out[37]: `array([1, 2])`

Consider the following array `v`:

In [38]:
```
# Create a numpy array

v = np.array([3, 2])
v
```

Out[38]: `array([3, 2])`

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

In [39]:
```
# Calculate the production of two numpy arrays

z = u * v
z
```

Out[39]: `array([3, 4])`

Dot Product

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

In [40]:
```
# Calculate the dot product

np.dot(u, v)
```

Out[40]: `7`

Adding Constant to a Numpy Array

Consider the following array:

In [41]:
```
# Create a constant to numpy array

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

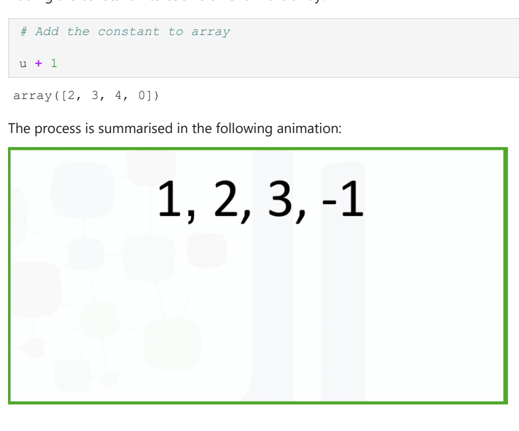
Out[41]: `array([ 1,  2,  3, -1])`

Adding the constant 1 to each element in the array:

In [42]:
```
# Add the constant to array

u + 1
```

Out[42]: `array([2, 3, 4, 0])`

The process is summarised in the following animation:



Mathematical Functions

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

In [43]:
```
# The value of pi

np.pi
```

Out[43]: `3.141592653589793`

We can create the following numpy array in Radians:

In [44]:
```
# 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:

In [45]:
```
# Calculate the sin of each elements

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

Out[45]: `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. We specify the starting point of the sequence and the ending point of the sequence. The parameter "num" indicates the Number of samples to generate, in this case 5:

In [46]:
```
# Makeup a numpy array within [-2, 2] and 5 elements

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

Out[46]: `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:

In [47]:
```
# Makeup a numpy array within [-2, 2] and 9 elements

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

Out[47]: `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 2π:

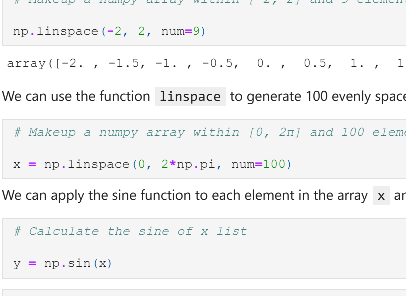
In [48]:
```
# Makeup a numpy array within [0, 2π] and 100 elements

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`:

In [49]:
```
# Calculate the sine of x list

y = np.sin(x)
```

In [50]:
```
# Plot the result

plt.plot(x, y)
```

Out[50]: `[<matplotlib.lines.Line2D at 0x28eef44a400>]`



Quiz on 1D Numpy Array

Implement the following vector subtraction in numpy: u-v

In [51]:
```
# Write your code below and press Shift+Enter to execute

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

Out[51]: `array([ 1, -1])`

▶ Click here for the solution

Multiply the numpy array z with -2:

In [52]:
```
# Write your code below and press Shift+Enter to execute

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

Out[52]: `array([-4, -8])`

▶ Click here for the solution

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

In [53]:
```
# Write your code below and press Shift+Enter to execute
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 0, 1, 0, 1])
x * y
```

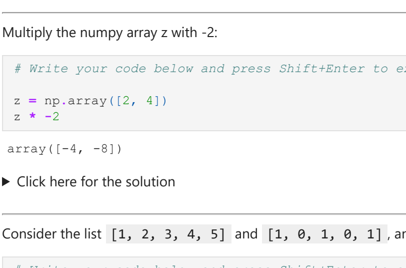
Out[53]: `array([1, 0, 3, 0, 5])`

▶ Click here for the solution

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 the dot product:

In [54]:
```
# Write your code below and press Shift+Enter to execute
a = np.array([-1, 1])
b = np.array([1, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a,b))
```

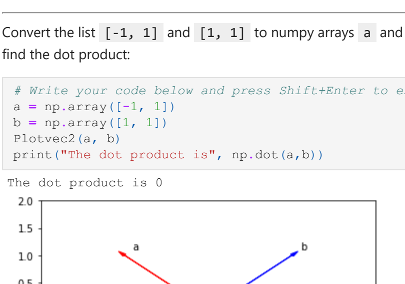
```
The dot product is 0
```



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 the dot product:

In [55]:
```
# Write your code below and press Shift+Enter to execute
a = np.array([1, 0])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a,b))
```

```
The dot product is 0
```

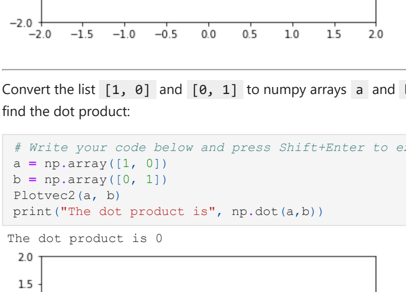


▶ Click here for the solution

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 the dot product:

In [56]:
```
# Write your code below and press Shift+Enter to execute
a = np.array([1, 1])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a,b))
```

```
The dot product is 1
```



▶ Click here for the solution

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.

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

▶ Click here for the solution

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow this article to learn how to share your work.

Author

Joseph Santarcangelo

Other contributors

Mavis Zhou

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-08-26	2.0	Lavanya	Moved lab to course repo in GitLab
