=Python Mod 5=

NUMPY 1D ARRAYS

The basics and array creations

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

0	1	2	3	4	Index

Index → a[0]:0 a[1]:1 a[2]:2 a[3]:3 a[4]:4 ← Value

A "numpy" array or "ndarray" is similar to a list
It's usually fixed in size and each element is of the same type (int, float, etc)

```
a: array([0,1,2,3,4])
type(a): numpy.ndarray
a.dtype: dtype('int64')
```

→ As numpy arrays contain data of the same type, we can use the attribute "dtype"

integer 64 bits

=Attributes=

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

0	1	2	3	4

```
a.size : 5
a.ndim : 1
a.shape : (5)
```

→ Number of elements
→ rank of the array
→ Tuple of integers indicating the size of the array in each dimension.

```
b=np.array([3.1, 11.02, 6.2])
type(b): numpy.ndarray
b.dtype: dtype('float64')
```

Indexing and Slicing

→ We can change the first element of the array or whatever element of the list.

```
c=np.array([20,1,2,3,4])
c: array([20,1,2,3,4])
c[0]=100
c: array([100,1,2,3,4])
c[4]=0
c: array([100,1,2,3,0])
```

→ Like list and tuples, we can slice a numpy array

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

0	1	2	3	4

```
d=c[1:4]
d: array([1,2,3])
```

→ Igual que en las listas el último elemento no se cuenta 4

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

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

0	1	2	3	4

```
c[3:5]=300,400
c: array([100,1,2,300,400])
```

BASIC OPERATIONS

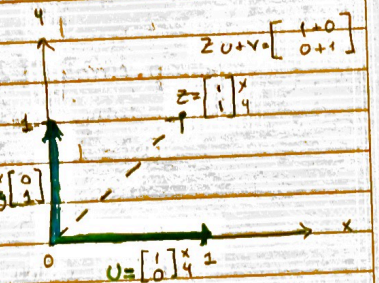
Vector addition and Subtraction

Euclidean Vectors

SUM

$$u = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad v = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$z = u + v = 4 \begin{bmatrix} 1+0 \\ 0+1 \end{bmatrix} = 4 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



✓ runs faster

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

× It takes more PC resources and time

```
u = [1,0]
v = [0,1]
z = []
for n, m in zip (u,v):
    z.append (n+m)
```

SUBSTRACTION

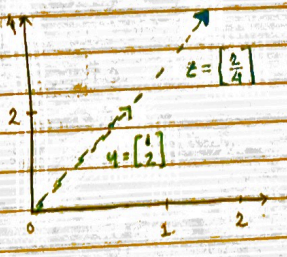
```
z = u - v
z.array ([1,-1])
```

```
z.append (n-m)
```

MULTIPLICATION WITH A SCALAR

$$y = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$z = 2y = \begin{bmatrix} 2(1) \\ 2(2) \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$




```
y = np.array ([1,2])
z = 2*y
z : array ([2,4])
```

```
y = [1,2]
z = []
for n in y:
    z.append (2*n)
```

1 + -1
1 - 1
0

0 / 1 - horizontal
0 - vertical

PRODUCT OF 2 NUMPY ARRAYS

$$u = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad v = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$z = u \circ v = \begin{bmatrix} 1*3 \\ 2*2 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

```
u = np.array ([1,2])
v = np.array ([3,2])
z = u * v
z : array ([3, 4])
```

```
u = [1,2]
v = [3,2]
z = []
for n, m in zip (u,v):
    z.append (n * m)
```

DOT PRODUCT

➔ The dot product is a single number given by the following term and represent how similar two vectors are.

$$u = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad v = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$u^T v = 1*3 + 2*1 = 5$$

```
u = np.array ([1,2])
v = np.array ([3,1])
result = np.dot (u,v)
result : 5
```

ADDING A CONSTANT TO A NUMPY ARRAY

```
u = np.array ([1,2,3,-1])
z = u + 1
z : array ([2,3,4,0])
```

1, 2, 3, -1

⇩

1+1, 2+1, 3+1, -1+1

UNIVERSAL FUNCTIONS

AVERAGE

```
a=np.array([1,-1,1,-1])
mean_a = a.mean()        # average of all elements
mean_a: 0.0
```

MAX VALUE

```
b=np.array([1,-2,3,4,5])
max_b=b.max()            # max value
max_b: 5
```

PLOT MATHEMATICAL FUNCTIONS

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

End Point; #'s of samples to generate; Interval; Start point

→ Returns evenly spaced #s over a specified interval

-2 -1 0 1 2

```
x=np.linspace(0,2*np.pi,100)
y=np.sin(x)
import matplotlib.pyplot as plt
% matplotlib inline
plt.plot(x,y)
```

2-DIMENSIONAL NUMPY ARRAYS

Numpy can be created with more than one dimension

The basics and array creation in 2D

```
a = [[11,12,13],[21,22,23],[31,32,33]]
A = np.array(a)
```

axis 1 (columns); axis 0 (rows)

A:
11 12 13
21 22 23
31 32 33

# of rows; # of columns

Attributes

A.ndim : 2 → axes or dimensions referred to as the rank ("ndim" → Number of nested lists)

A.shape : (3,3) Number of lists = shape not ndim

(size of nested lists (# of columns); # rows)

A.size : 9 3×3 = 9

→ We can use rectangular brackets to access the different elements of the array

A: [[A[0][0], A[0][1], A[0][2]], [A[1][0], A[1][1], A[1][2]], [A[2][0], A[2][1], A[2][2]]]

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]

- Rows A[1][2] : 23
- Colums A[0][0] : 11

SLICING

```
A[0,0:2]
```

Igual que en listas el último elemento no se cuenta

first row; across 2 colums (0,1)

array([11,12])

```
A[0:2,2] : array([13,23])
```

across 2 rows (0,1)

Norma

BASIC OPERATIONS

SUM

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X + Y = \begin{bmatrix} 1+2 & 0+1 \\ 0+1 & 1+2 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

```
X= np.array([[1,0],[0,1]])
Y= np.array([[2,1],[1,2]])
Z = X+Y
Z :array ([[3,1],
           [1,3]])
```

MULTIPLICATION WITH A SCALAR

$$4 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$2 * 4 = \begin{bmatrix} 2*2 & 2*1 \\ 2*1 & 2*2 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

```
4=np.array([[2,1],[1,2]])
Z=2*4;
Z.array([[4,2],
         [2,4]])
```

MULTIPLICATION OF 2 ARRAYS #HADAMARD PRODUCT

Multiply each of the elements in the same position

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X * 4 = \begin{bmatrix} (1)(2) & (0)(1) \\ (0)(1) & (1)(2) \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

```
X= np.array([[1,0],[0,1]])
4=np.array([[2,1],[1,2]])
Z= X*4;
Z :array ([[2,0],
           [0,2]])
```

MATRIX MULTIPLICATION . DOT

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix}$$

3

In linear algebra before we can multiply matrix "A" by matrix "B" we must make sure that the # of columns in matrix A is equal to the # of rows in matrix B

$0 + 1 + (-1)$

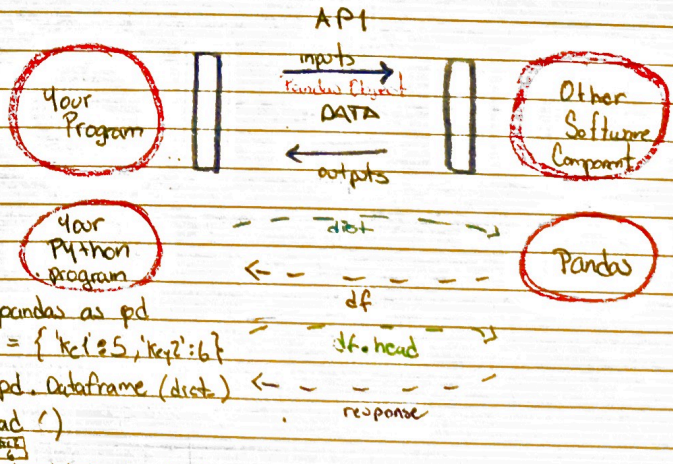
$(0)(1)+(1)(1)+(1)(1) = 0$
$(0)(1)+(1)(1)+(1)(1) = 2$
$(1)(1)+(0)(1)+(1)(-1) = 0$
$(1)(1)+(0)(1)+(1)(1) = 2$

$$AB = \begin{bmatrix} 0 & 2 \\ 0 & 2 \end{bmatrix}$$

```
A = np.array([[0,1,1],[1,0,1]])
B= np.array([[1,1],[1,1],[-1,1]])
C= np.dot(A,B);
C: array([[0,0]],
         [0,2]])
```

SIMPLE APIs - Application Program Interface

Two pieces of software talk to each other




```
import pandas as pd
dict_ = {'key1':5,'key2':6}
df = pd.Dataframe(dict_)
df.head()
```

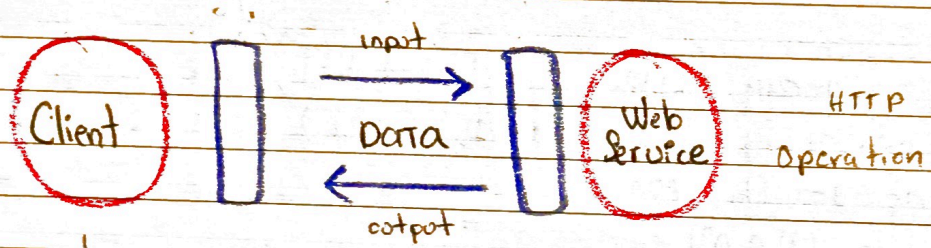

Norma

REST APIs

RE presentational State Transfer

allow to connect to the internet → help take advantage of resorces like
- storage
- AI algorithms

- REST APIs are used to interact with web services,
  ie Apps that you call through the internet
- They have a set of Rules regarding:
  1. Communication
  2. Input or Request
  3. Output or Response



my program
my code is the client