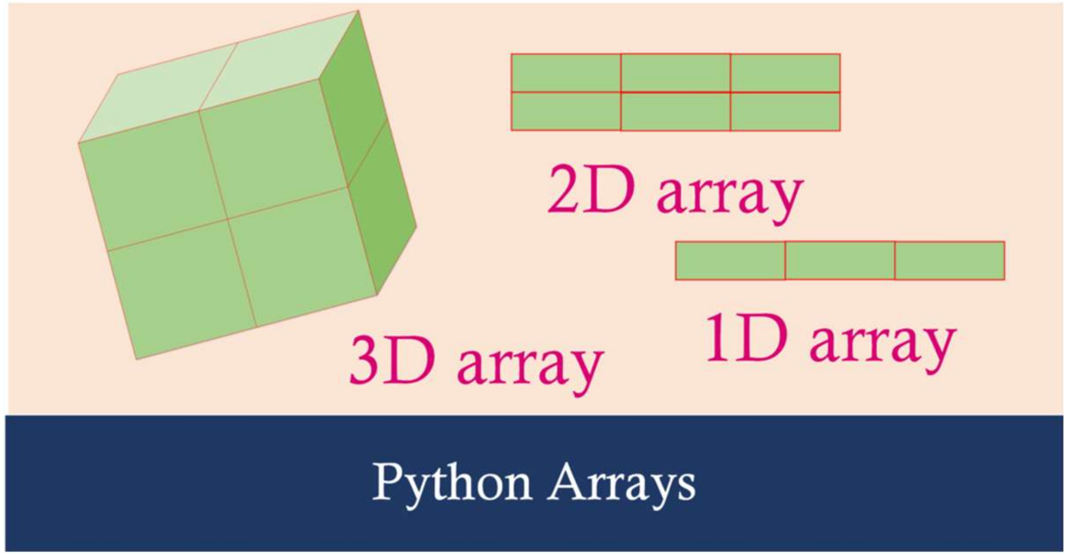
Arrays

Lecture 4: Arrays in python



- **M.Sc. Riyadh Seed Agid**
- **Salahaddin University – Erbil**
- **riyadh.agid@su.edu.krd**

Arrays in python

In many algorithms, data can be represented mathematically as a vector or a matrix.

The basic object in NumPy is the array

An array is a systematic arrangement of objects (usually numbers) in rows and columns or it is a list that contain mixed datatypes. An empty list can be utilized **by []** and **append** command can be used to merge data to the end of the list.

1-D Arrays

Example

```
>>> import numpy as np

# Create a 1-D array by passing a list into NumPy's array() function.
>>> np.array([8, 4, 6, 0, 2])
array([8, 4, 6, 0, 2])

# The string representation has no commas or an array() label.
>>> x = np.array([1, 3, 5, 7, 9])
>>>print(x)
[1  3   5  7  9]
```

2-D Arrays

These are often used to represent matrix or 2nd order tensors.

```
import numpy as np

>>> x = np.array([[1, 2, 3], [4, 5, 6]])

>>> print(x)
[[1 2 3]
 [4 5 6]]
```

3-D arrays

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

>>> x = np.array([[[1, 2, 3], [4, 5, 6]],[[1, 2, 3], [4, 5, 6]]])

>>> print(x)
[[[1 2 3]
  [4 5 6]]

  [[1 2 3]
  [4 5 6]]]
```

Example

Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
```

	Result
`print(a.ndim)`	0
`print(b.ndim)`	1
`print(c.ndim)`	2
`print(d.ndim)`	3

list

Example

```
>>>x = [“python”,” programming”]
>>>x.append (“physics”)
>>>print x
[‘python’, ‘programming’,’physics’]
```

Append a single element that will extend the list:

If you want to extend more than one element you should use extend, because you can only append one element or one list of element:

Example

```
>>> x = [1,2]
>>>x.extend ([3,4 5 ,6,7])
>>>print x
[1,2,3,4,5,6,7]
```

❑ An array can be defined by one of the four procedures

1) arange	The arguments of NumPy arange() that define the values contained in the array correspond to the numeric parameters start, stop, and step.
2) ones	This is an array where each and every element is one.
3) zeros	This array contains nothing but zeros.
4) linspace	Numpy linspace function returns an evenly spaced sequence of numbers for a given interval.

1) Arange works exactly the same as range, but produces an array instead of a list

Example

```
>>> import numpy as np

>>>np.arange(10,0,-2)
>>>print x
[10  8  6  4  2]
```

Start = 10

Stop = 0

Step = -2

In this example, start is 10. Therefore, the first element of the obtained array is 10. step is -2, which is why your second value is 10-2, that is 8, while the third value in the array is 8-2, which equals 6.

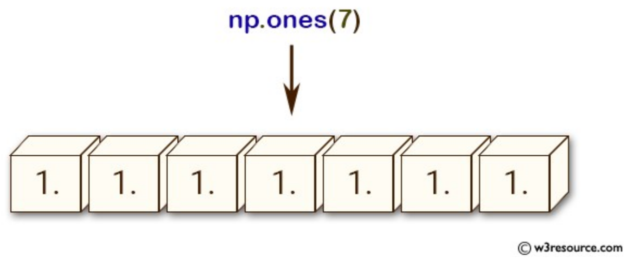
```
>>> import numpy as np

>>> np.arange(start=1, stop=10, step=3)
array([1, 4, 7])
```

NOTE: The value of **stop** is not included in an array.

2) Ones similarly creates an array of a certain size containing all ones.

Example

```
>>>x = ones(7)
>>>print x
[1. 1. 1. 1. 1. 1. 1.]
```



3) The zeros() function is used to get a new array of given shape and type, filled with zeros .

Example: numpy.zeros() function

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

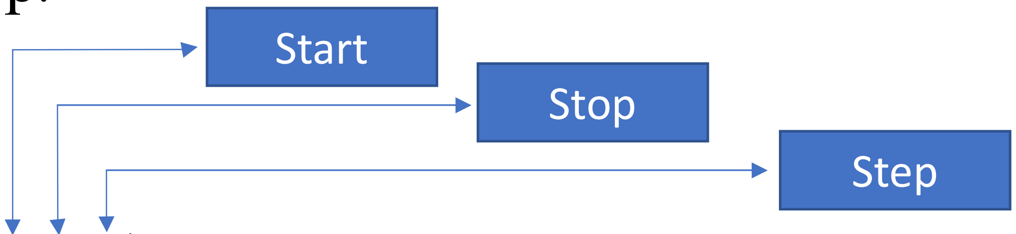
Rows → 3, Columns → 2

```
>>> x = zeros(4)
>>>print x
[0. 0. 0. 0. 0.]
```

4) **linspace:** linspace(start, stop, n=50)

Producing an array of n points evenly spaced between starting point and the stop.

Example 1:




```
>>>np.linspace(2,3,5)
 array([2, 2.25, 2.5, 2.75, 3.] )
```

NOTE: The value of **stop** is included in an array.

Example 2:

```
>>> np.linspace(2,3,5, endpoint=False)
array([2, 2.2, 2.4, 2.6, 2.8])
```

Example 3:

```
>>> np.linspace(2,3, 5, retstep=True)
(array([2,2.25,2.5,2.75,3],0.25)
```

Note: In general, we will select start point < stop so we can get an array where the entries are increasing in value. On the other hand, you can choose start = stop (to obtain an array with every value equal) or start > stop (to display an array with values decrease).

Example :

```
>>> np.linspace(2,10,4)
array([ 2.        ,  4.66666667,  7.33333333, 10.        ])
```

Example :

```
>>> np.linspace(2, 2, 5)
>>>array([2, 2, 2, 2, 2])
```

Example :

```
>>> np.linspace (3, 2, 5)
>>> array([3.  2.75  2.5  2.25  2])
```

An arrays in general are the main structure used in **NumPy.** Therefore all of the array manipulations and creations techniques that are available here are contained in the NumPy package.

I= [a, b, c, d, ….] defines list, I, of numbers.
x= array([a, b, c, d,….]) defines an array, x

To produce an array of float numbers just we need to add **float** to the end.
x= array([a, b, c, d, ….], dtype=float)

Example 1:

```
>>> x=array([1,2,3,4,5])
>>> print x
[1 2 3 4 5]
```

Example 2:

```
>>> x= array ([1, 2, 3, 4, 5], dtype = float)
>>> print x
[1. 2. 3. 4. 5.]
```

Operations with array

The calculation technique that we introduced in the last week can be applied on array as well. Every week we will be understanding an extra methods of manipulating and calculating with arrays so it is important to be compatible with the basic ideas. For now, we will just regard an array as a list of numbers. Afterward, we will see how the methods we introduce here can be applied to vector calculations, making graphs, and analysing our experimental data.

Lets start with these two arrays

```
>>> x= array([2, 1, 3, 5, 4], dtype= float)
>>> y= array([3, 1, 6, 2, 7], dtype= float)
```

1-Addition and subtraction of a constant.

```
>>> z =x+5
>>>print z
[7.  6.  8.  10.  9. ]
>>> z= y-2
>>>print z
[1.  -1. 4.  0.  5. ]
```

2- Addition and subtraction of arrays

```
>>> z= x+y
>>> print (z)
[5.  2.  9.  7.  11.]
>>> z=x-y
>>>print (z)
[-1.  0.  -3.  3.  -3. ]
```

Important note:** In case of arrays operation of different shapes or sizes, you will see the error message ***ValueError: shape mismatch: objects cannot be broadcast to a single shape.

Example

```
>>>x= array([1,3,5,6])
>>>y= array([0,1, 2 ])
>>> z= x+y
```

This will not work

3-Multiplication and division by a constant

It is also possible to multiply or divide an array by a constant. This multiplies or divides each element in the array by the specified value.

Example

```
>>> x= array([2, 1, 3, 5, 4], dtype= float)
>>> y= array([3, 1, 6, 2, 7], dtype= float)
>>> z=5*x
>>>print z
[10.  5.  15.  25.  20.]
 >>> z= x/2.0
>>>print z
[1.  0.5  1.5  2.5  2. ]
```

4- Multiplication and division of arrays

Multiplication and division between arrays is possible only if they have same lengths

Example

```
>>> z= x*y
>>>print z
[6.  1.  18.  10.  28]
```

Example

>>>z=x/y

>>>print z

[0.6666667 1. 0.5   2.5   0.57142857]

5- Power and roots

When we intend to raise an array x to the power n. Every element in the array is raised to the same specified power.

Example

Let

>>>i= array([2,3,4])

>>>j= i**2

>>>print j

array([4,9,16])

>>>sqrt(i)

array([1.414, 1.732 2.])

Vector algebra

Some of the operations that we introduced in the previous sections are also applicable to vectors (such as addition and subtraction).

```
>>> a = array([1, 3, 5], dtype =float)
>>> b = array([4, 2, 6], dtype=float)
```

We can introduce 3D vector for each as

a = i+3j+5k

b = 4i+2j+6k

Addition and subtraction of vectors can be done :

c = a+b

c = a-b

Also we can change or re-scale a vector if we multiply it by a constant such as:

c = 5.3*a

Dot and cross product of vectors

There are functions in **NumPy** ready to find out the **dot** and **cross** products between vectors. If **a** and **b** are two vectors, then the dot and cross products denoted in python as **dot(a,b)** and **cross (a,b).**

Example

```
>>>dot (a,b)
40.0
>>> cross(a,b)
array([8., 14., -10.])
```

Also the value (magnitude) of a vector can be obtained (which is the square root of the sum of the squares of each element) by:

```
>>>linalg.norm(a)
5.916
```