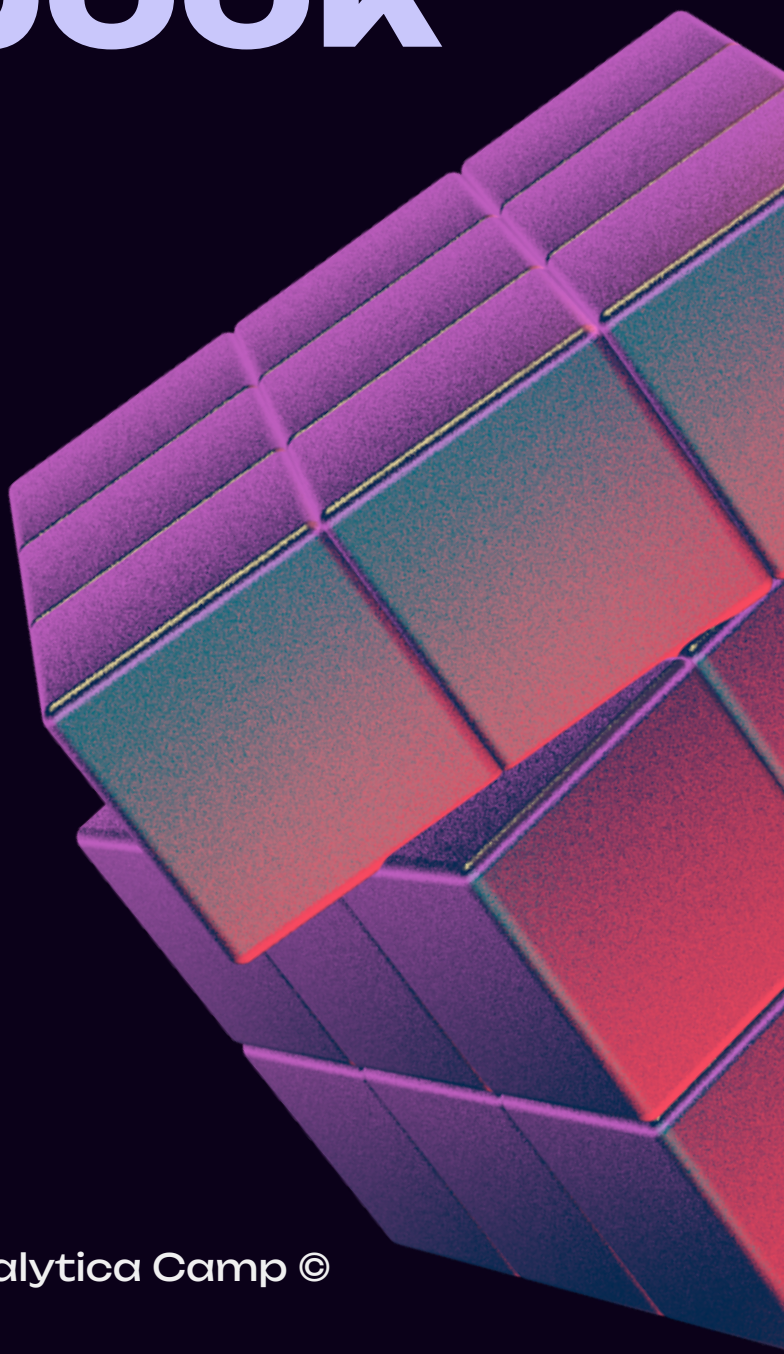
analytica camp

# NumPy handbook

PART 1



By Orian Ay, Analytica Camp ©

# “NumPy” stands for Numerical Python.

**The definition on the documentation:**

"NumPy (Numerical Python) is an open source Python library used in almost every field of science and engineering. It’s the universal standard for working with numerical data in Python. It is at the core of the scientific Python and PyData ecosystems. "

Basically, NumPy is used for array operations in Python. Most of these operations are implemented on C.

Although Python has its own built-in data type ( lists ) for the representations of Arrays, it is too slow to work with. (50x slower than NumPy)

## Let's recap what an array is

Array is a data structure where values or items are placed in a linear order, which means the memory assigned to each item is contiguous.

Arrays are used to store multiple values with the same data type in a single variable, instead of declaring separate variables for each value. Arrays are widely used in data science.

**The reason for Numpy being faster is because it stores arrays in a continuous place in memory and it can access the same set of memory for a particular time.**

(Read more : Locality of Reference)

But also insertion and deletion operations become costly as data is stored in contiguous memory locations.

Numpy is an essential part of the Data Science, Machine Learning Projects. It has a wide usage in Computer vision for image processing, in NLP for vectorization.

# How to install NumpPy

If you don't have Python installed, consider installing Anaconda, as you don't need to install NumPy separately with this distribution.

- For exploratory and interactive computing, you can use Jupyter Notebook or Jupyter Lab after installing Anaconda.

- For writing scripts and packages you can use Spyder or Visual Studio Code.

Each of these is installed together with the Anaconda distribution.

**If you already have Python, you can simply install Numpy by using one of these commands:**

**- pip install numpy**
**- conda install numpy**

## How to import NumPy

```
>>> import numpy as np
```

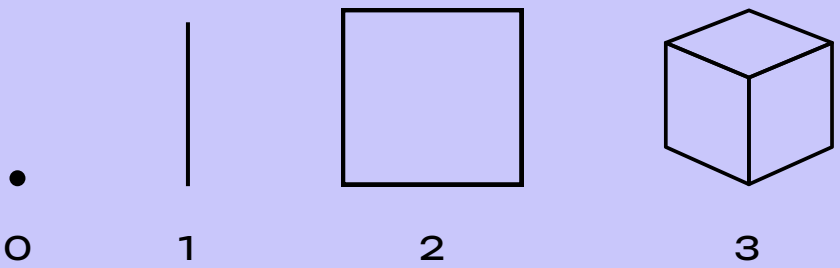
"Numpy" is shortened to "np" for better readability of code using NumPy. This is a widely adopted convention that you should follow so that anyone working with your code can easily understand it.

## How to initialize an array

```
>>> x = np.array( [ 1, 2, 3 ] )
```

For initializing the 1-Dimensional array we can simply give it a list

Geometric Representation of Dimensions:




Dimensions

analytica camp

For 2-Dimensional array we give a list in a list:

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

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

We can initialize as many dimensions as we want

The way to initialize the array is using list comprehension. If you are not familiar with list comprehension, look at the basic example below:

```
>>>  squares = [ i * i for i in range ( 10 ) ]

>>>  squares
[ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 ]
```

List comprehensions are often described as being more pythonic than loops. Because it is simple and easier to read and understand.

In the example above, we defined the list and its contents at the same time.

## Initializing array using list comprehension

```
>>> x = np.array ( [ [ [ i + j + k for k in range ( 5 ) ]
                for j in range ( 4 ) ] for i in range ( 3 ) ] )
>>> x
array( [ [ [0, 1, 2, 3, 4],
           [1, 2, 3, 4, 5],
           [2, 3, 4, 5, 6],
           [3, 4, 5, 6, 7] ],

         [ [1, 2, 3, 4, 5],
           [2, 3, 4, 5, 6],
           [3, 4, 5, 6, 7],
           [4, 5, 6, 7, 8] ],

         [ [2, 3, 4, 5, 6],
           [3, 4, 5, 6, 7],
           [4, 5, 6, 7, 8],
           [5, 6, 7, 8, 9] ] ] )
```

# There are few properties we can examine to inspect an array

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

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

### Number of dimensions / axes

```
>>> x.ndim
2
```

### Size of each dimensions

```
>>> x.shape
(2, 3)
```

### Total size of array

```
>>> x.size
6
```

### Array data type

```
>>> x.dtype
dtype('int64')
```

analytica camp

# Other ways to create an array

**Creating an array of given shape and type, filled with 0 s:**

```
>>> np.zeros ( ( 2, 10 ) )

array( [ [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.] ] )
```

**Creating an array of given shape and type, filled with 1s:**

```
>>> np.ones ((3, 3), dtype=np. int64)

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

**Creating an array of given interval, filled with evenly spaced values, allows you to specify the size of the steps:**

```
>>> np.arange( 0, 20, 2 )

 array( [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ] )
```

**Creating an array of given interval, filled with evenly spaced values, allows you to specify the number of the steps:**

```
>>> np.linspace(0, 10, 4)

   array([ 0.  ,  3.33333333 ,  6.66666667 , 10.  ])
```

analytica camp

# Array Indexing & Slicing

Array indexing is the same as accessing an array element.

```
>>> x = np.array ( [ [ i for i in range (10, 15) ],
                            [ i for i in range (100, 105) ] ] )

array( [ [ 10,  11,  12,  13,  14 ],
           [ 100, 101, 102, 103, 104 ] ] )


>>> x[ 0 ]

array([10, 11, 12, 13, 14])

 >>> x[ 1 ]

array([100, 101, 102, 103, 104])

>>>x[0, 3]
13

>>>x[1, 2]

 102
```

```
>>>x[0, : ]

array([10, 11, 12, 13, 14])


>>>x[:, 0]

array([ 10, 100])


>>>x[:, 3]

array([ 13, 103])


>>>x[:, 3:7]

array( [ [ 13,  14],
         [103, 104 ] ] )
```

analytica camp

# Fancy Indexing

Fancy indexing is a method used when working in arrays. It is an advanced form of simple indexing. An index is used to represent the position of an element within a given array. Fancy indexing is used to get multiple elements by passing a list of indices. It's not a part of vanilla Python.

```
>>>x[ : , : : -1]

array ( [ [ 14,  13,  12,  11,  10 ],
          [ 104, 103, 102, 101, 100 ] ] )

>>>x[ : : -1, :]

array ( [ [ 100, 101, 102, 103, 104],
          [ 10,  11,  12,  13,  14 ] ] )

>>>x[ [ 0, 1, 0, 1 ], [ 0, 1, 2, 3 ] ]

array ( [ 10, 101,  12, 103 ] )

>>>x [ : , [ 0, 1, 4, 3 ] ]

array ( [ [ 10,  11,  14,  13 ],
          [ 100, 101, 104, 103 ] ] )
```

analytica camp

# Array Reshaping

```
>>> x = np.arange 9)
```

```
>>> x
array ([O, 1, 2, 3, 4, 5, 6, 7, 8])
```

**We have created 1-Dimensional array.**
**For reshaping an array to 3-dimensional:**

```
>>> x.reshape ( (3, 3) )
```

```
array( [ [O, 1, 2],
         [3, 4, 5],
         [6, 7, 8] ] )
```

**We can reshape an array as long as the elements required for reshaping are equal in both shapes.**

```
>>> x. reshape ( (5, 5) )
```

```
Traceback (most recent call last):
File "‹stdin>", line 1, in <module>
ValueError: cannot reshape array of size 9 into shape (5,5)
```

analytica camp

# Elementwise operations

```
>>>x = np.arange (10)

>>>y = np.arange (10, 20) [ : : -1]

>>>x
array ( [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] )

>>>y
array ( [19, 18, 17, 16, 15, 14, 13, 12, 11, 10] )

>>>x + y
array( [19, 19, 19, 19, 19, 19, 19, 19, 19, 19] )

>>>x - y
array( [-19, -17, -15, -13, -11,  -9,  -7,  -5,  -3,  -1] )


>>>x*y
array([ 0, 18, 34, 48, 60, 70, 78, 84, 88, 90])
```

# Math functions

```
>>>x = np.linspace(0, 5, 4)

array([0.   , 1.66666667,    3.33333333,    5.    ])

>>>np.sqrt(x)

array([0.   , 1.29099445,
           1.82574186,   2.23606798])

>>>np.log(x)

array([  -inf,    0.51082562,
           1.2039728 , 1.60943791])

>>>np.sin(x)

array([ 0.   ,  0.99540796,
           -0.19056796, -0.95892427])
```

**For more math functions see official documentation:**

numpy.org/doc/stable/reference/routines.math.html

- For the Part 2,
  Follow Orian Ay

analytica camp