Unit - 5

NumPy

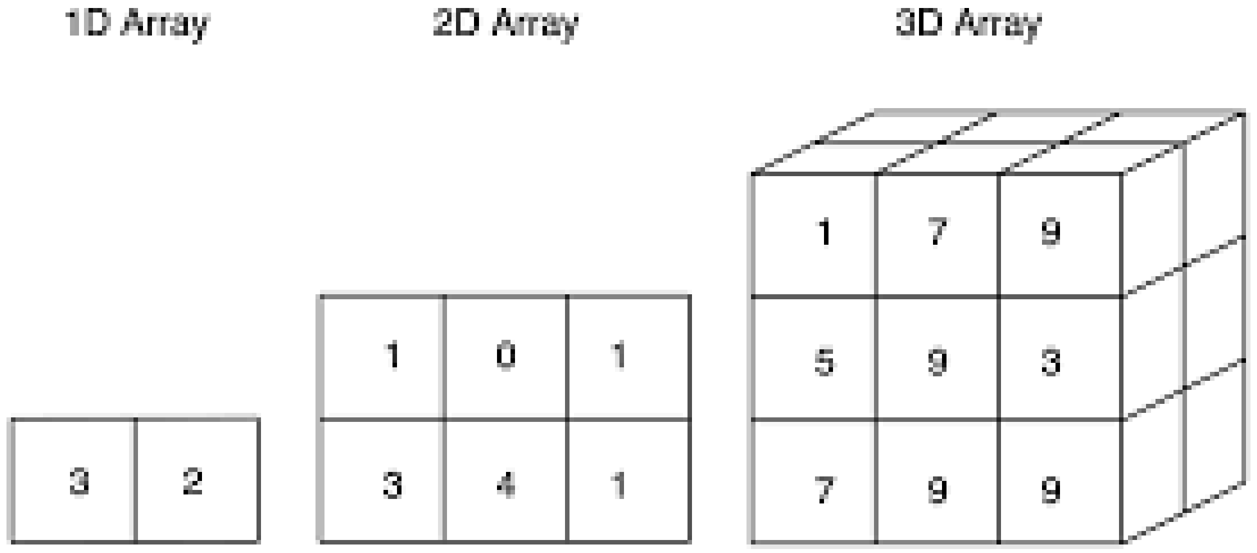
NumPy Package

- NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays.
- Using NumPy, mathematical and logical operations on arrays can be performed.
- NumPy is a Python package. It stands for ‘Numerical Python’.
- It is a library consisting of multidimensional array objects and a collection of routines for processing of array.
- **Numeric**, the ancestor of NumPy, was developed by Jim Hugunin.
- Another package Numarray was also developed, having some additional functionalities.
- In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open-source project.

- It is easy to integrate with C/C++ and Fortran.
- For any scientific project, NumPy is the tool to know. It has been built to work with the N-dimensional array, linear algebra, random number, Fourier transform, etc.

Operations using NumPy

- Using NumPy, a developer can perform the following operations –
- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.
- **NumPy Array:** Numpy array is a powerful N-dimensional array object which is in the form of rows and columns.
- We can initialize numpy arrays from nested Python lists and access it elements.


1D Array
2D Array
3D Array
3
2
1
0
1
3
4
1
1
7
9
5
9
3
7
9
9

- NumPy is often used along with packages like **SciPy** (Scientific Python) and **Matplotlib** (plotting library).
- This combination is widely used as a replacement for MatLab, a popular platform for technical computing.
- However, Python alternative to MatLab is now seen as a more modern and complete programming language.
- It is open-source, which is an added advantage of NumPy.

- The most important object defined in NumPy is an N-dimensional array type called **ndarray**.
- It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.
- Every item in a ndarray takes the same size as the block in the memory.
- Each element in ndarray is an object of the data-type object (called **dtype**).
- Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types.

Why to use NumPy?

- NumPy is memory efficient, meaning it can handle the vast amount of data more accessible than any other library.
- Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping.
- On top of that, NumPy is fast.
- In fact, TensorFlow and Scikit learn to use NumPy array to compute the matrix multiplication in the back end.

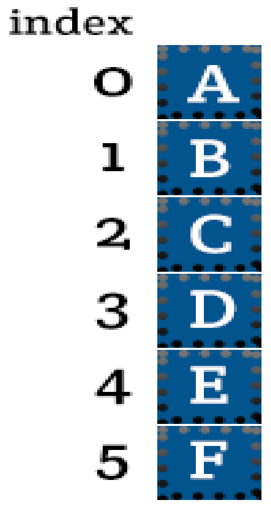
List vs. NumPy

- List are slow as compared to NumPy.

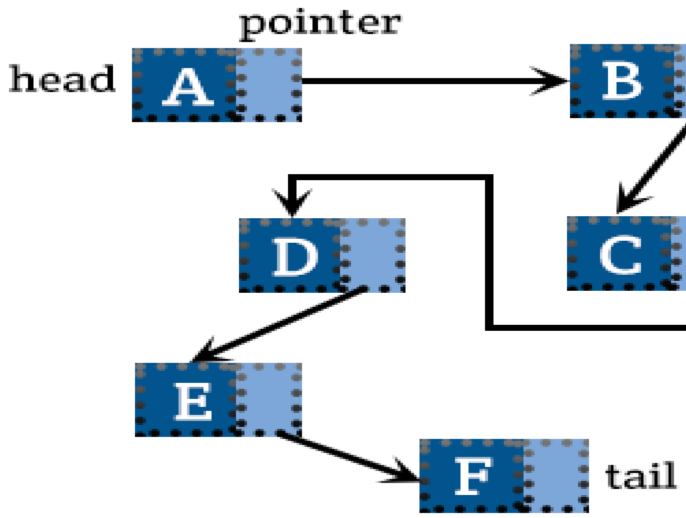
(In NumPy we can specify int32,int16 or int8, in list we store value with four details size, reference count, object type and object value).

- No type checking when iterating through objects.
- NumPy uses contiguous memory allocation.
- Uses SIMD Vector Processing.
- Effective cache utilization.
- More operations can be performed in NumPy.

Array



Linked List



How to install NumPy

- You can install NumPy **using Anaconda:**
  - **conda install -c anaconda numpy**
- In **Jupyter Notebook** :
  - **import sys !conda install --yes --prefix {sys.prefix} numpy**
- Or else
  - **pip install numpy**

Import NumPy and Check Version

- The command to import numpy is
- **import numpy as np**
- Above code renames the NumPy namespace to np. This permits us to prefix Numpy function, methods, and attributes with " np " instead of typing " numpy."
- It is the standard shortcut you will find in the numpy literature
- To check your installed version of Numpy use the command
- **np.__version__**

Creating the NumPy Array

- NumPy arrays are a bit like Python lists, but still very much different at the same time.
- As the name kind of gives away, a NumPy array is a central data structure of the numpy library.
- An instance of ndarray class can be constructed by different array creation routines described later in the tutorial. The basic ndarray is created using an array function in NumPy as follows-

- **numpy.array**

- It creates a ndarray from any object exposing an array interface, or from any method that returns an array.
- **numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)**
- The **ndarray** object consists of a **contiguous** one-dimensional segment of computer memory, combined with an indexing scheme that maps each item to a location in the memory block.

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array or any (nested) sequence.
2 3	**dtype** The desired data type of array, optional**copy**Optional. By default (true), the object is copied. If we do not define the data type, then it will determine the type as the minimum type which will require to hold the object in the sequence.
4	**order**The order parameter specifies the memory layout of the array. When the object is not an array, the newly created array will be in C order (row head or row-major) unless 'F' is specified. When F is specified, it will be in Fortran order (column head or column-major)

5	**subok** By default, returned array forced to be a base class array. If true, sub-classes passed through
6	**ndmin** Specifies minimum dimensions of the resultant array

- The difference between C and F is just whether the array is row major or column major (i.e. either row or column entries are stored in adjacent memory address). **C order means that operating row-rise on the array will be slightly quicker. F order means that column-wise operations will be faster**.

order	no copy	copy=True
'K'	Unchanged	F and C order preserved.
'A'	Unchanged	When the input is F and not C then F order otherwise C order
'C'	C order	C order
'F'	F order	F order

- **Finding the dimensions of the Array**
- The **ndim** function can be used to find the dimensions of the array.
- **Finding the size of each array element**
- The **itemsize** function is used to get the size of each array item. It returns the number of bytes taken by each array element.
- **Finding the data type of each array item**
- To check the data type of each array item, the **dtype** function is used. Consider the following example to check the data type of the array items.

- **Finding the shape and size of the array**
- To get the shape and size of the array, the size and shape function associated with the numpy array is used.

Creating Array in NumPy

- **Numpy.empty() in Python**
- As the name specifies, The empty routine is used to create an uninitialized array of specified shape and data type.
- Return a new array of given shape and type, with random values.
- The syntax is given below.
- **numpy.empty(shape, dtype = float, order = 'C')**
- It accepts the following parameters.

- **Shape:** The desired shape of the specified array.
- **dtype:** The data type of the array items. The default is the float.
- **Order:** The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order.

Example:-

```
import numpy as np
arr = np.empty((3,2), dtype = int)
print(arr)
```

- **NumPy.Zeros**
- This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 0.
- The syntax is given below.

numpy.zeros(shape, dtype = float, order = 'C')

- It accepts the following parameters.
- **Shape:** The desired shape of the specified array.
- **dtype:** The data type of the array items. The default is the float.
- **Order:** The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order.

Example:-

```
import numpy as np
arr = np.zeros((3,2), dtype = int)
print(arr)
```

- **NumPy.ones**
- This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 1.
- The syntax to use this module is given below.

numpy.ones(shape, dtype = none, order = 'C')

- It accepts the following parameters.

- **Shape:** The desired shape of the specified array.
- **dtype:** The data type of the array items.
- **Order:** The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order

Example:-

```
import numpy as np
arr = np.ones((3,2), dtype = int)
print(arr)
```

Numpy.Full():-

- The **numpy.full() function** is used to return a new array of a given shape and data type filled with fill_value.
- **Syntax**
- **numpy.full(shape, fill_value, dtype=None, like=None)**
- **Parameters**
- The numpy.full() function takes the following parameter values:

•**shape**: This represents the shape of the desired array.
•**fill_value**: This represents the fill value.
•**dtype**: This represents the data type of the desired array. This is an optional parameter.
•**like**: This represents the prototype or the array_like object.

Return value

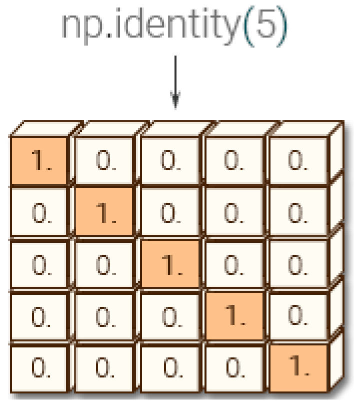
The numpy.full() function returns an array of fill_value shape, data type, and order.

Example:-

```
import numpy as np
# implementing the numpy.full() function
myarray = np.full((2, 2), 5)
print(myarray)
```

Numpy.identity() :-

The identity array is a square array with ones on the main diagonal. The identity() function return the identity array



Syntax:
numpy.identity(n, dtype=None)

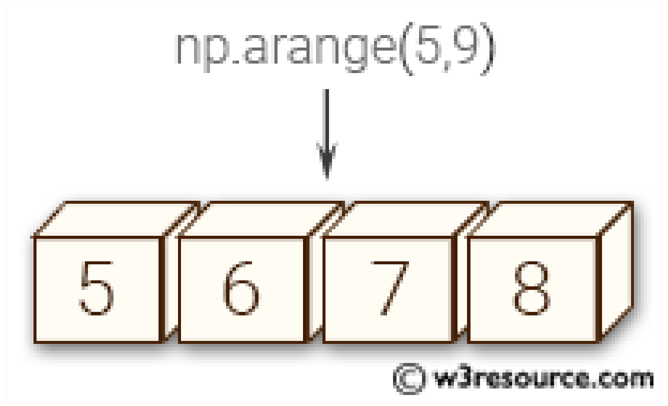
Name	**Description**	**Required / Optional**
n	Number of rows (and columns) in n x n output.	Required
dtype	Data-type of the output. Defaults to float.	optional

Example:-

```
import numpy as np
np.identity(2)
```

numpy.arange() function

- The arange() function is used to get evenly spaced values within a given interval
- Values are generated within the half-open interval [start, stop].
- For integer arguments the function is equivalent to the Python built-in range function, but returns an ndarray rather than a list.
- When using a non-integer step, such as 0.1, the results will often not be consistent.
- It is better to use linspace for these cases.
- **Syntax:**
- **numpy.arange([start,]stop, [step,]dtype=None)**


np.arange(5,9)
5
6
7
8
© w3resource.com

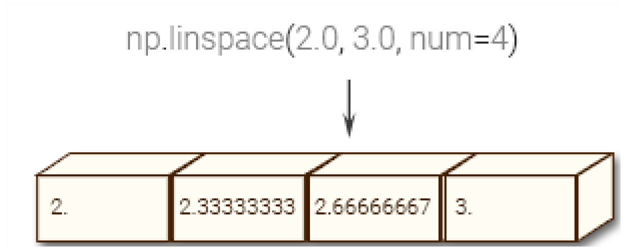
Name	Description	Required / Optional
start	Start of interval. The interval includes this value. The default start value is 0.	Optional
stop	End of interval. The interval does not include this value, except in some cases where step is not an integer and floating point round-off affects the length of out.	Required
step	Spacing between values. For any output out, this is the distance between two adjacent values, out[i+1] - out[i]. The default step size is 1. If step is specified as a position argument, start must also be given.	Optional
dtytpe	The type of the output array. If dtype is not given, infer the data type from the other input arguments.	Optional

- **Return value:**
- arange : ndarray - Array of evenly spaced values. For floating point arguments, the length of the result is ceil((stop - start)/step). Because of floating point overflow, this rule may result in the last element of out being greater than stop.
- **Example:-**

```
import numpy as np
arr1 = np.arange(5)
arr2 = np.arange(5,9)
arr3 = np.arange(5,9,3)
print(arr1,arr2,arr3)
```

numpy.linspace() function

- The linspace() function returns evenly spaced numbers over a specified interval [start, stop].The endpoint of the interval can optionally be excluded.
- **Syntax:**
- **numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)**



Name	**Description**	**Required / Optional**
start	The starting value of the sequence.	Required
stop	The end value of the sequence, unless endpoint is set to False. In that case, the sequence consists of all but the last of num + 1 evenly spaced samples, so that stop is excluded. Note that the step size changes when endpoint is False.	Required
num	Number of samples to generate. Default is 50. Must be non-negative.	Optional
endpoint	If True, stop is the last sample. Otherwise, it is not included. Default is True.	Optional
retstep	If True, return (samples, step), where step is the spacing between samples.	Optional
dtype	The type of the output array. If dtype is not given, infer the data type from the other input arguments	Optional

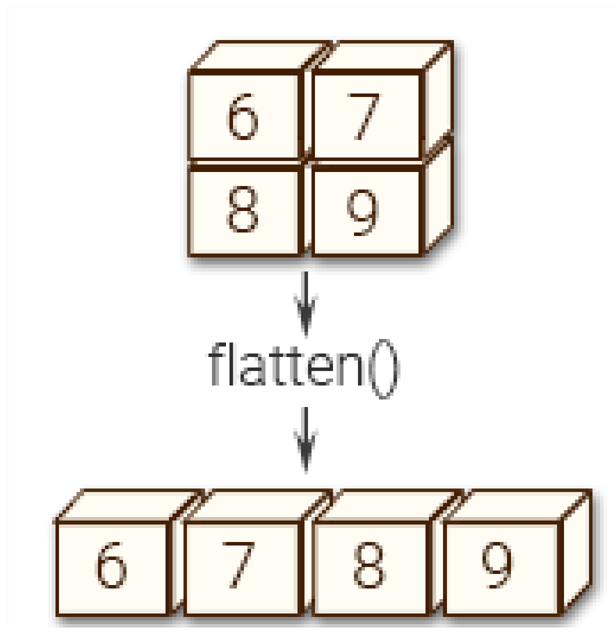
- **Return value:**
- ndarray - There are num equally spaced samples in the closed interval [start, stop] or the half-open interval [start, stop) (depending on whether endpoint is True or False).
- step : float, optional - Only returned if retstep is True
  Size of spacing between samples.
- **Example:-**

```
import numpy as np
arr1 = np.linspace(3.0, 4.0, num=7)
print(arr1)
arr2 = np.linspace(3.0,4.0, num=7, endpoint=False)
print(arr2)
arr3 = np.linspace(3.0,4.0, num=7, retstep=True)
print(arr3)
```

- In simple terms arange returns values based on step size and linspace relies on sample size.
- **arange** – Return values with in a range which has a space between values (in other words the step).
- **linspace** – Return set of samples with in a given interval.

numpy.ndarray.flatten() function

- The flatten() function is used to get a copy of an given array collapsed into one dimension.
- **Syntax:**
- **ndarray.flatten(order='C')**



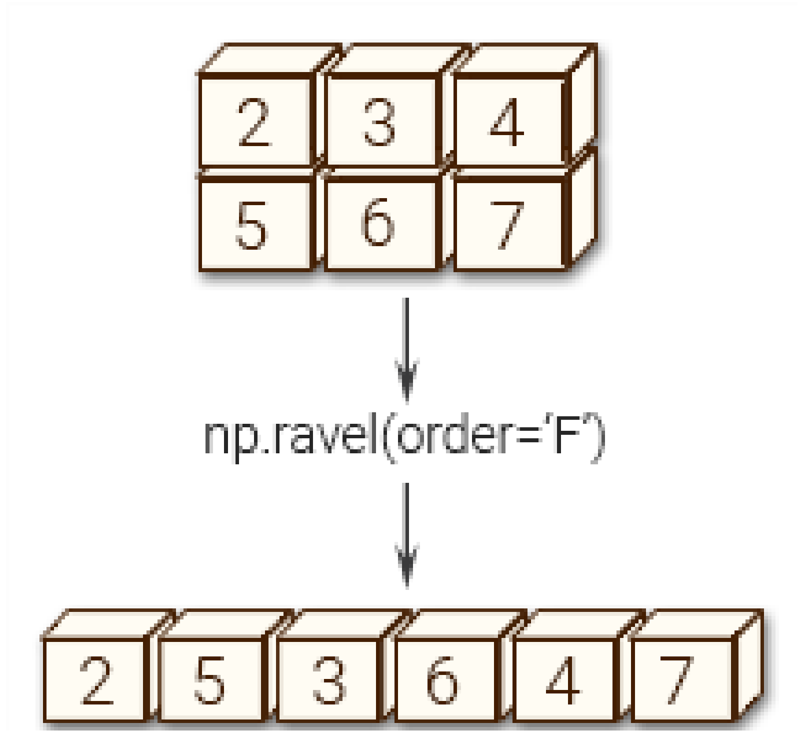
Name	**Description**	**Required / Optional**
order	‘C’ means to flatten in row-major (C-style) order. ‘F’ means to flatten in column-major (Fortran- style) order.	Optional

- **Return value:**
- ndarray - A copy of the input array, flattened to one dimension.
- **Example:-**

```
import numpy as np
y = np.array([[2,3], [4,5]])
print(y)
y.flatten()
```

numpy.ravel() function

- The ravel() function is used to create a contiguous flattened array.
- A 1-D array, containing the elements of the input, is returned. A copy is made only if needed.
- As of NumPy 1.10, the returned array will have the same type as the input array. (for example, a masked array will be returned for a masked array input)
- **Syntax:**
- **numpy.ravel(a, order='C')**


2
3
4
5
6
7
np.ravel(order='F')
2
5
3
6
4
7

Name	**Description**	**Required / Optional**
a	Input array. The elements in a are read in the order specified by order, and packed as a 1-D array.	Required
order	‘C’ means to flatten in row-major (C-style) order. ‘F’ means to flatten in column-major (Fortran-style) order.	Optional

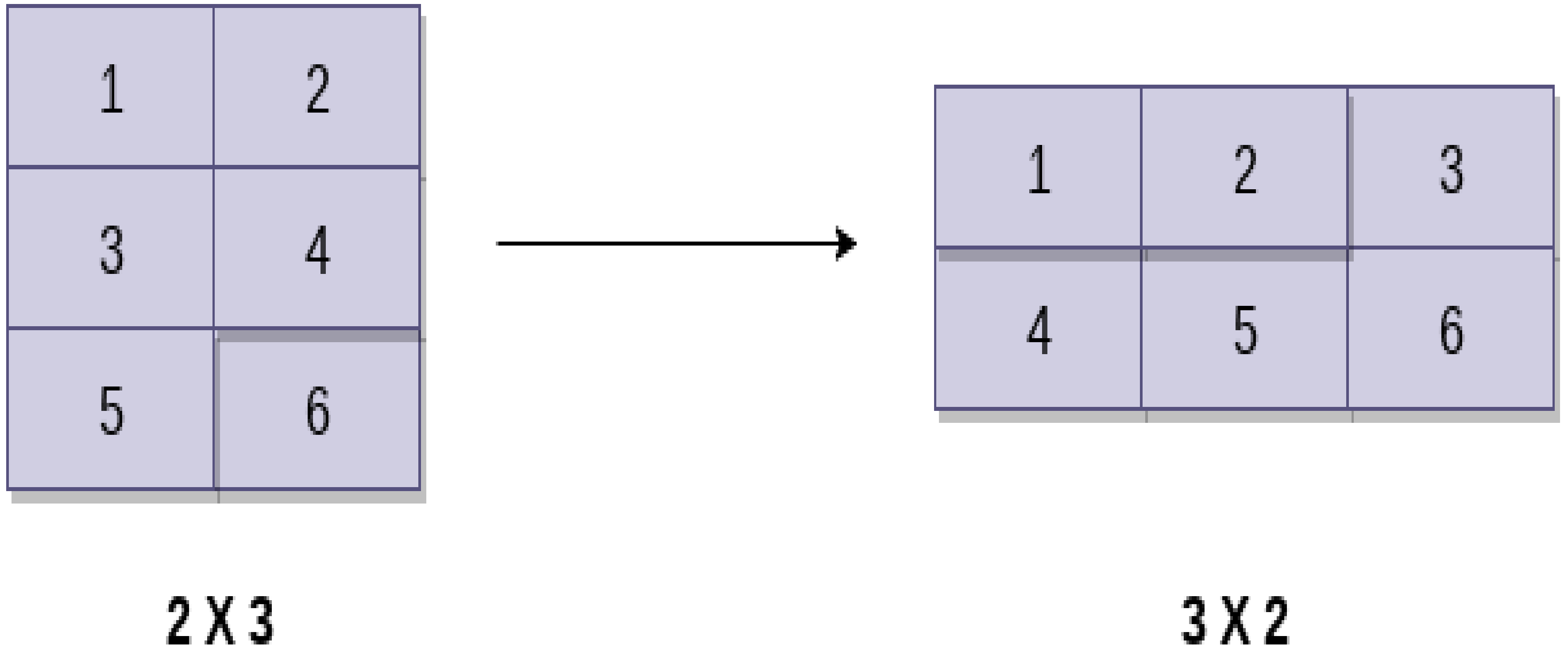
Example:-

```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6]])
print(x)
print(np.ravel(x))
```

- **Differences between Flatten() and Ravel()**
- **a.ravel()**:
  (i) Return only reference/view of original array
  (ii) If you modify the array you would notice that the value of original array also changes.
  (iii) Ravel is faster than flatten() as it does not occupy any memory.
  (iv) Ravel is a library-level function.
- **a.flatten()** :
  (i) Return copy of original array
  (ii) If you modify any value of this array value of original array is not affected.
  (iii) Flatten() is comparatively slower than ravel() as it occupies memory.
  (iv) Flatten is a method of an ndarray object.

Reshaping the array objects

- By the shape of the array, we mean the number of rows and columns of a multi-dimensional array.
- However, the numpy module provides us the way to reshape the array by changing the number of rows and columns of the multi-dimensional array.
- The reshape() function associated with the ndarray object is used to reshape the array. It accepts the two parameters indicating the row and columns of the new shape of the array.


1
2
3
4
5
6
2 X 3
1
2
3
4
5
6
3 X 2

- **Syntax: numpy.reshape(array, shape, order = 'C')**
- **array :** [array_like]Input array
- **shape :** [int or tuples of int] e.g. if we are aranging an array with 10 elements then shaping it like numpy.reshape(4, 8) is wrong; we can do numpy.reshape(2, 5) or (5, 2)
- **order :** C order means that operating row-rise on the array will be slightly quicker FORTRAN-contiguous order in memory (first index varies the fastest). F order means that column-wise operations will be faster. ‘A’ means to read / write the elements in Fortran-like index order if, array is Fortran contiguous in memory, C-like order otherwise

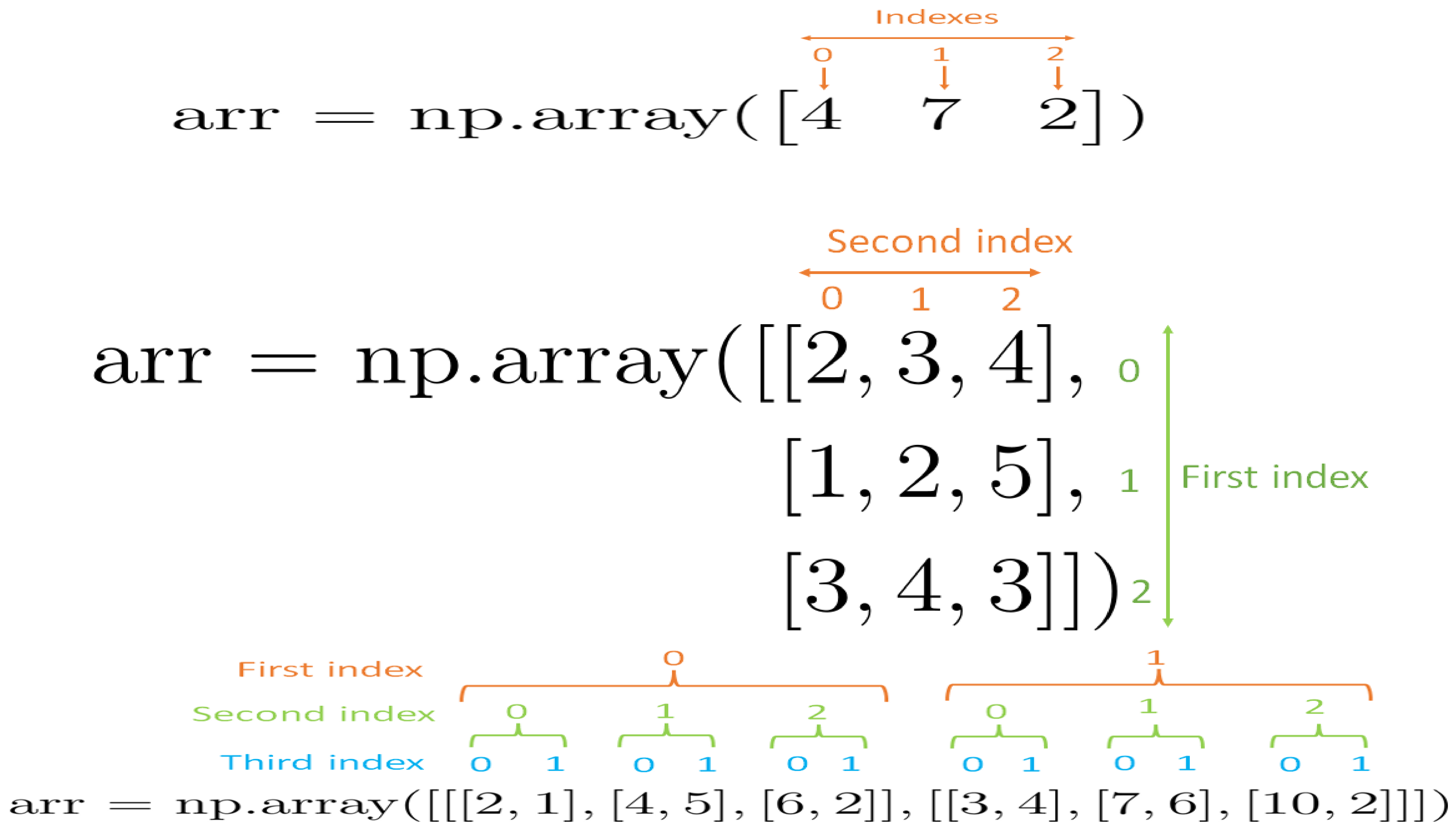
- **Example:-**

```
#reshaping the array
import numpy as np
a = np.array([[1,2],[3,4],[5,6]])
print("printing the original array..")
print(a)
a=a.reshape(2,3)
print("printing the reshaped array..")
print(a)
```

Slicing in the Array

- Slicing in the NumPy array is the way to extract a range of elements from an array.
- Slicing in the array is performed in the same way as it is performed in the python list.
- **Example:-**

```
import numpy as np
a = np.array([[1,2],[3,4],[5,6]])
print(a[0,1])
print(a[2,0])
```

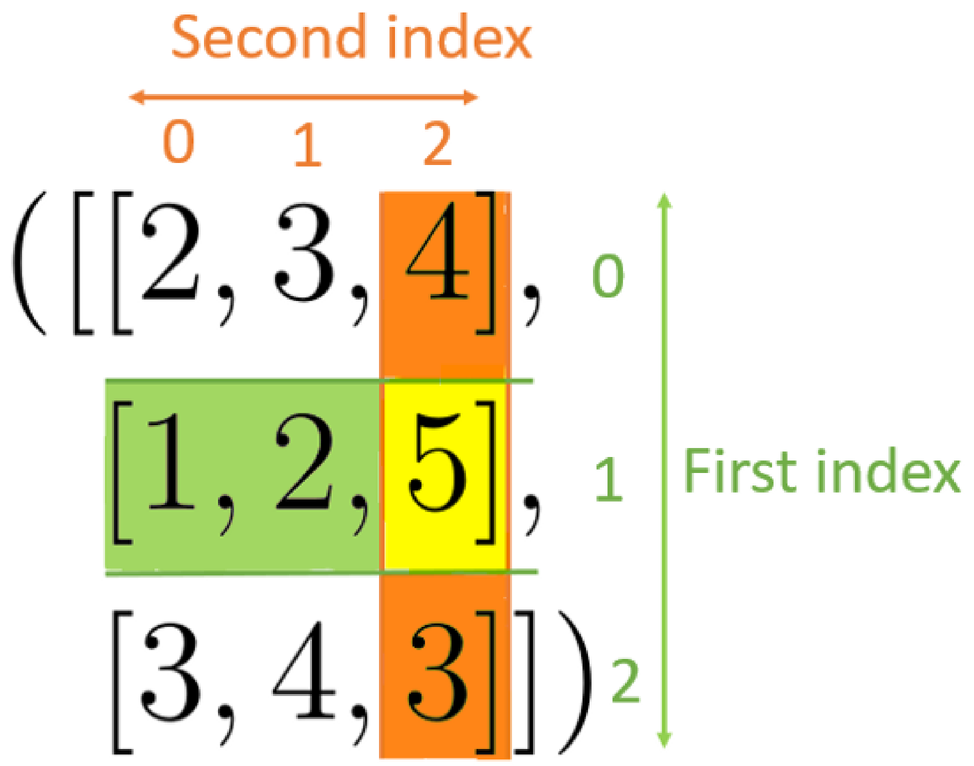


```
arr = np.array([4   7   2])
```

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

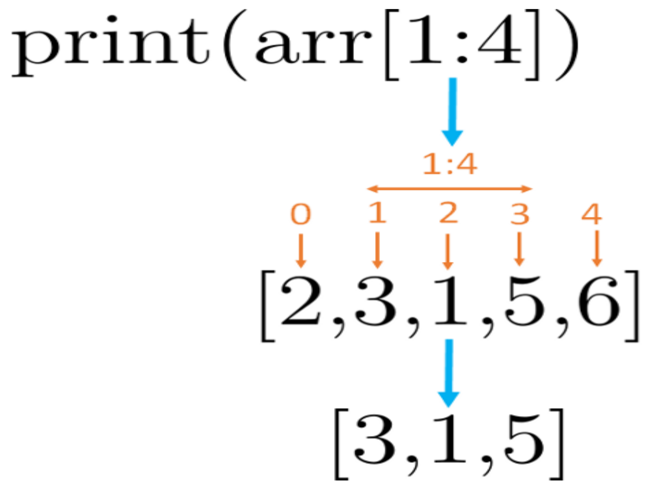
```
arr = np.array([[[2, 1], [4, 5], [6, 2]], [[3, 4], [7, 6], [10, 2]]])
```

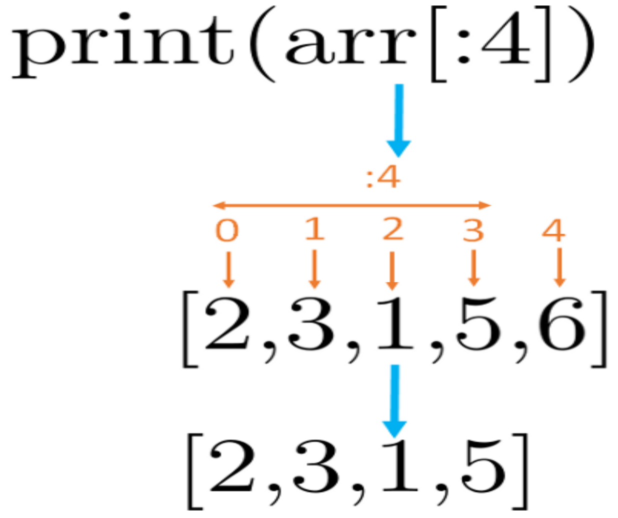
$\text{print}(\text{arr}[1, 2])$

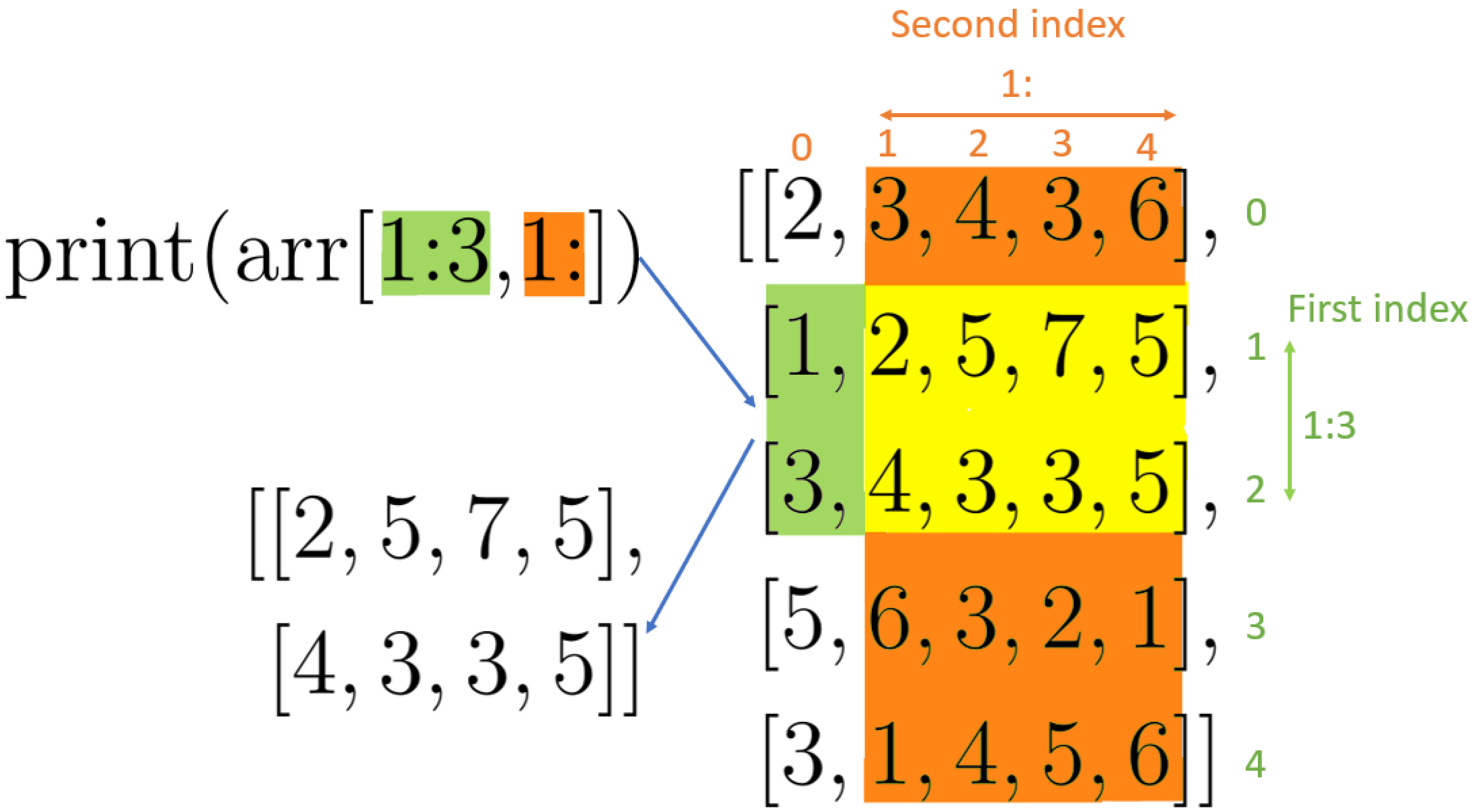


$$\begin{array}{lccc} & \multicolumn{3}{c}{\overleftrightarrow{\text{Indexes}}} \\ & -3 & -2 & -1 \\ & \downarrow & \downarrow & \downarrow \\ \text{arr} = \text{np.array}(& [4 & 7 & 2]) \end{array}$$

$$\begin{array}{lcccll} & \multicolumn{3}{c}{\overleftrightarrow{\text{Second index}}} & & \\ & -3 & -2 & -1 & & \\ \text{arr} = \text{np.array}(& [[2, & 3, & 4], & -3 & \\ & [1, & 2, & 5], & -2 & \updownarrow \text{ First index} \\ & [3, & 4, & 3]]) & -1 & \end{array}$$


print(arr[1:4])
1:4
0 1 2 3 4
[2,3,1,5,6]
[3,1,5]


print(arr[:4])
:4
0 1 2 3 4
[2,3,1,5,6]
[2,3,1,5]



```
print(arr[1:3, 1:])
```

```
[[2, 5, 7, 5],
 [4, 3, 3, 5]]
```

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

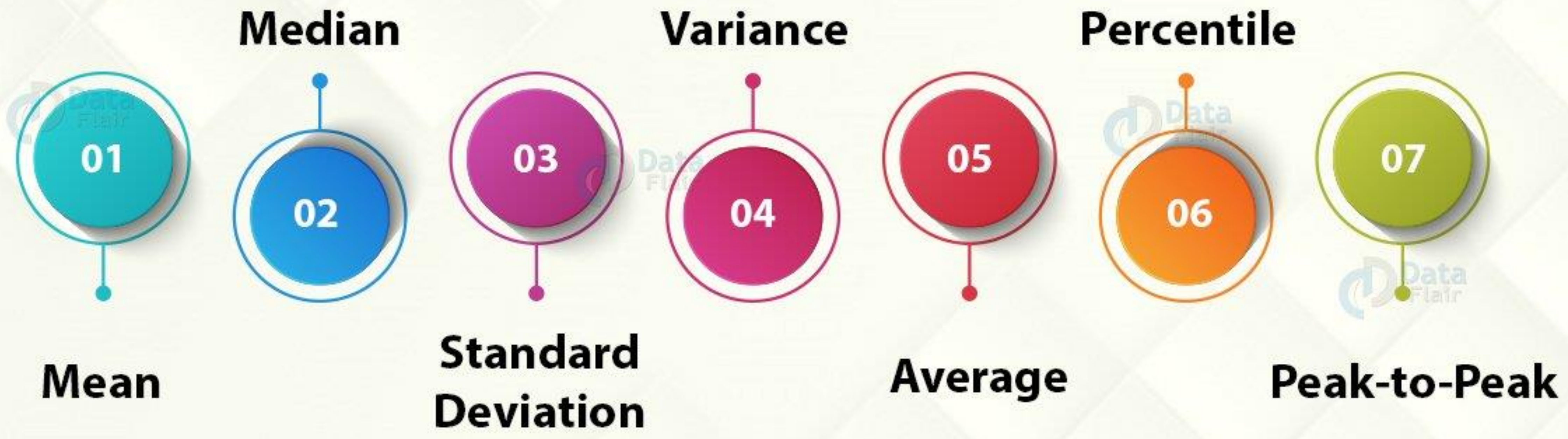
- We pass slice instead of index like this: [*start*:*end*].
- We can also define the step, like this: [*start*:*end*:*step*].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1
- **Example:-**

-
  ```
  import numpy as np

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

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

Statical functions in numpy

- Statistics is concerned with collecting and then analysing that data.
- It includes methods for collecting the samples, describing the data, and then concluding that data.
- NumPy is the fundamental package for scientific calculations and hence goes hand-in-hand for NumPy statistical Functions.
- NumPy contains various statistical functions that are used to perform statistical data analysis.
- These statistical functions are useful when finding a maximum or minimum of elements. It is also used to find basic statistical concepts like standard deviation, variance, etc.

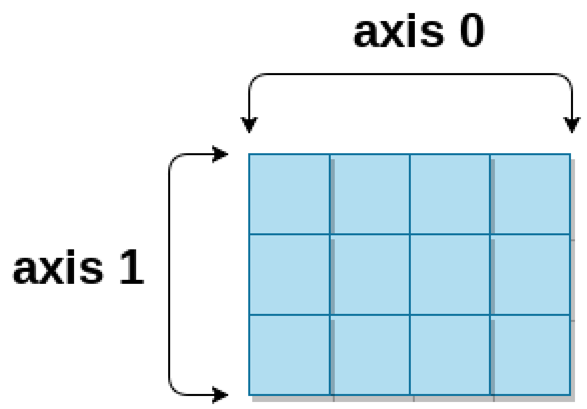
NumPy Statistical Functions



- **1. np.amin()-** This function determines the minimum value of the element along a specified axis.
  2. np.amax()- This function determines the maximum value of the element along a specified axis.
  3. np.mean()- It determines the mean value of the data set.
  4. np.median()- It determines the median value of the data set.
  5. np.std()- It determines the standard deviation
  6. np.var – It determines the variance.
  7. np.ptp()- It returns a range of values along an axis.
  8. np.average()- It determines the weighted average
  9. np.percentile()- It determines the nth percentile of data along the specified axis.

NumPy Array Axis

- A NumPy multi-dimensional array is represented by the axis where axis-0 represents the columns and axis-1 represents the rows.
- We can mention the axis to perform row-level or column-level calculations like the addition of row or column elements.



NumPy Array

- To calculate the maximum element among each column, the minimum element among each row, and the addition of all the row elements.

numpy.min()

- Numpy min() function is used to get a minimum value along a specified axis

Syntax of Numpy.min()

```
np.min(a, axis=None)
```

- a parameter refers to the array on which you want to apply np.min() function.
- axis parameter is optional and helps us to specify the axis on which we want to find the minimum values

- **Example:-**

```
import numpy as np
a = np.array([[50, 15, 89, 23, 64], [45, 98, 25, 17, 55], [35, 37, 9, 100, 
61]])
print(a)
print('Minimum value in arr: ', np.min(a))
print('Minimum value in arr along axis 0: ', np.min(a, axis=0))
print('Minimum value in arr along axis 1: ', np.min(a, axis=1))
```

numpy.max()

- Numpy max() function is used to get a maximum value along a specified axis.
- **Syntax of Numpy.max()**
- **np.max(a, axis=None**)
- a parameter refers to the array on which you want to apply np.max() function.
- axis parameter is optional and helps us to specify the axis on which we want to find the maximum values.

Example:-

```
import numpy as np
a = np.array([[50, 15, 89, 23, 64], [45, 98, 25, 17, 55], [35, 37, 9, 100,
61]])
print(a)
#maximum function in numpy
print('Maximum value in arr: ', np.max(a))
print('Maximum value in arr along axis 0: ', np.max(a, axis=0))
print('Maximum value in arr along axis 1: ', np.max(a, axis=1))
```

Numpy sum() in Python | np sum() in Python

- Numpy sum() function is used to sum the elements of Numpy array over the provided axis.
- Syntax of Numpy sum()

```
        np.sum(a, axis=None, dtype=None)
```

- a parameter is used to specify the array upon which we have to perform sum() function.
- axis parameter is optional and is used to specify the axis along which we want to perform addition. When you do not provide axis parameter value to np.sum() function then it adds all the values together and produces a single value.
- dtype parameter controls the data type of the resultant sum.

- **Example:-**

```
import numpy as np
a = np.array([32, 4, 8, 12, 20])
print('Sum of arr is ', np.sum(a))
```

numpy.mean()

- Numpy mean() function is used to calculate arithmetic mean of the values along the specified axis
- Syntax of Numpy mean()

```
        np.mean(a, axis=None, dtype=None)
```

- Here, a parameter is used to pass Numpy array to mean() function.
- axis parameter is optional and is used to specify the axis along which we want to perform arithmetic mean. When no axis value is passed then arithmetic mean of entire value is computed and a single value is returned.
- dtype parameter controls the data type of the resultant mean

Example:-

```
import numpy as np
a = np.array([[32, 4, 8, 12, 20], [35, 5, 15, 10, 30]])
print('Mean of arr is ', np.mean(a))
print('Mean of arr along axis0 is ', np.mean(a, axis=0))
print('Mean of arr along axis1 is ', np.mean(a, axis=1))
```

numpy.average() in Python

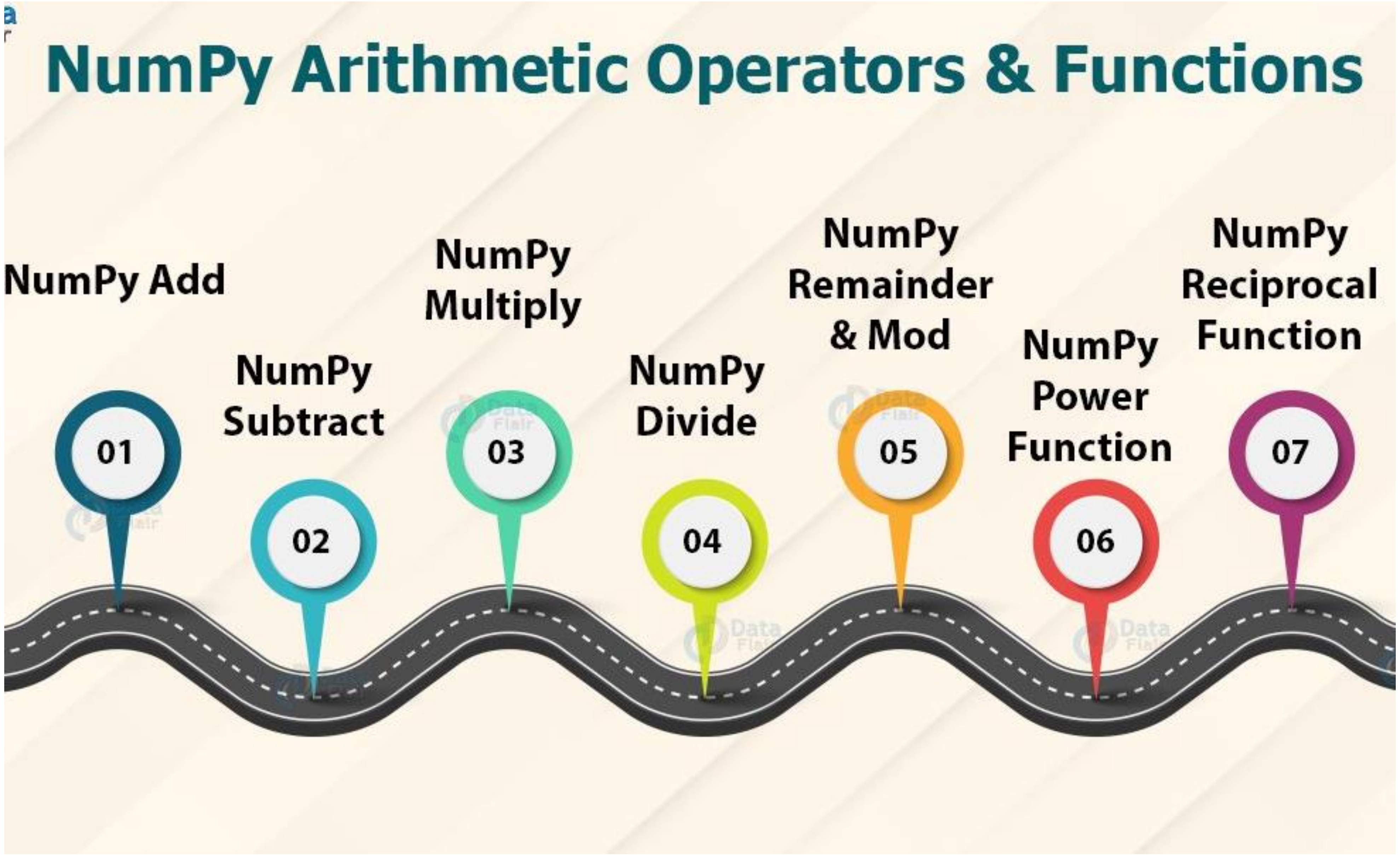
- Numpy average() is used to calculate the weighted average along the specified axis
- **Syntax of Numpy average()**

```
np.average(arr, axis=None, weights=None)
```

- Here arr refers to the array whose weighted average is to be calculated.
- axis parameter is optional and is used to specify the axis along which we want to perform a weighted average. When no axis value is passed then a weighted average of entire values is computed and a single value is returned.
- weights parameter is optional and is used to specify the weight for the values present in arr. When no value is passed for weights parameter then the weight is considered to be one for each value. weights parameter takes the value in the form of Numpy array or list.

NumPy Arithmetic Operations

- Arithmetic operations are possible only if the array has the same structure and dimensions.
- We carry out the operations following the rules of array manipulation.
- We have both functions and operators to perform these functions.



NumPy Arithmetic Operators & Functions

numpy.add()

- **numpy.add()** function is used when we want to compute the addition of two array.
- It add arguments element-wise.
- **Syntax :** numpy.add(arr1, arr2, /, out=None, *, where=True, casting=’same_kind’, order=’K’, dtype=None, subok=True[, signature, extobj], ufunc ‘add’)
- **Parameters :**
  arr1 : [array_like or scalar] Input array.
  arr2 : [array_like or scalar] Input array.
  out : [ndarray, optional] A location into which the result is stored.
    -> If provided, it must have a shape that the inputs broadcast to.
    -> If not provided or None, a freshly-allocated array is returned.

where : [array_like, optional] Values of True indicate to calculate the ufunc at that position, values of False indicate to leave the value in the output alone.
****kwargs :**Allows to pass keyword variable length of argument to a function. Used when we want to handle named argument in a function.
Return : [ndarray or scalar] The sum of arr1 and arr2, element-wise. Returns a scalar if both arr1 and arr2 are scalars.