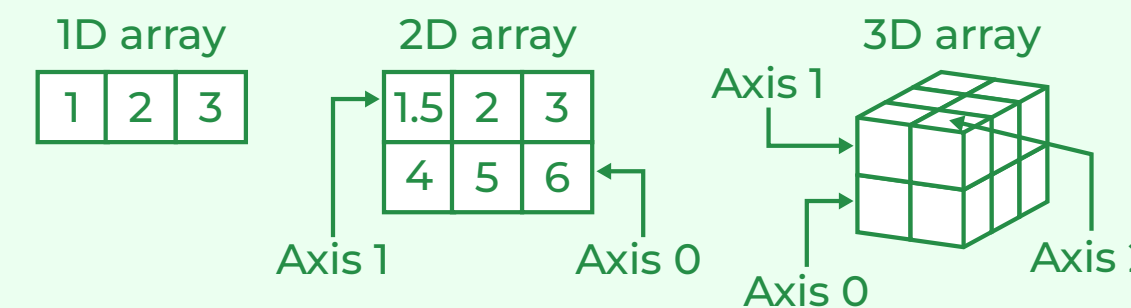


# NumPy Cheat Sheet

## NumPy stands for Numerical Python.

It is one of the most important foundational packages for numerical computing & data analysis in Python. Most computational packages providing scientific functionality use NumPy's array objects as the lingua franca for data exchange.

## Types of Numpy Array



## Creating Arrays Commands

### One Dimensional Array

From Python List	<code>np.array([1, 2, 3, 4, 5])</code>
From Python Tuple	<code>np.array((1, 2, 3, 4, 5))</code>
fromiter() function	<code>np.fromiter((a for a in range(8)), float)</code>

### Python3

- create a NumPy array from a list  
`li = [1, 2, 3, 4]`  
`print(np.array(li))`
- create a NumPy array from a tuple  
`tup = (5, 6, 7, 8)`  
`print(np.array(tup))`
- create a NumPy array using fromiter()  
`iterable = (a for a in range(8))`  
`print(np.fromiter(iterable, float))`

### Multi-Dimensional Array

Using Python Lists	<code>np.array([[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12]])</code>
Using empty()	<code>np.empty([4, 3], dtype=int)</code>

### Python3

- create a NumPy array from a list  
`list_1 = [1, 2, 3, 4]`  
`list_2 = [5, 6, 7, 8]`  
`list_3 = [9, 10, 11, 12]`  
`print(np.array([list_1, list_2, list_3]))`

- create a NumPy array using `numpy.empty()`  
`print(np.empty([4, 3], dtype=int))`

## Initial Placeholders

### One Dimensional Array

<code>arange()</code>	<code>np.arange(1, 10)</code>
<code>linspace()</code>	<code>np.linspace(1, 10, 3)</code>
<code>zeros()</code>	<code>np.zeros(5, dtype=int)</code>
<code>ones()</code>	<code>np.ones(5, dtype=int)</code>
<code>random.rand()</code>	<code>np.random.rand(5)</code>
<code>random.randint()</code>	<code>np.random.randint(5, size=10)</code>

### Python3

- create a NumPy array using `numpy.arange()`  
`print(np.arange(1, 10))`
- create a NumPy array using `numpy.linspace()`  
`print(np.linspace(1, 10, 3))`
- create a NumPy array using `numpy.zeros()`  
`print(np.zeros(5, dtype=int))`
- create a NumPy array using `numpy.ones()`  
`print(np.ones(5, dtype=int))`
- create a NumPy array using `numpy.random.rand()`  
`print(np.random.rand(5))`
- create a NumPy array using `numpy.random.randint()`  
`print(np.random.randint(5, size=10))`

### N-dimensional Numpy Arrays

<code>zeros()</code>	<code>np.zeros([4, 3], dtype = np.int32)</code>
<code>ones()</code>	<code>np.ones([4, 3], dtype = np.int32)</code>
<code>full()</code>	<code>np.full([2, 2], 67, dtype = int)</code>
<code>eye()</code>	<code>np.eye(4)</code>

### Python3

- create a NumPy array using `numpy.zeros()`  
`print(np.zeros(1, 10))`
- create a NumPy array using `numpy.ones()`  
`print(np.ones([4, 3], dtype = np.int32))`
- create a NumPy array using `numpy.full()`  
`print(np.full([2, 2], 67, dtype = int))`
- create a NumPy array using `numpy.eye()`  
`print(np.eye(4))`

## Inspecting Properties

Size	<code>arr.size</code>
Length	<code>len(arr)</code>
Shape	<code>arr.shape</code>
Datatype	<code>arr.dtype</code>
Changing Datatype of Array	<code>arr.astype('float64')</code>
Converting Array to List	<code>arr.tolist()</code>

## Saving and Loading File

Saving array on disk	<code>np.save("file", np.arange(5))</code>
Loading a file	<code>np.load("file.npy")</code>
Importing a Text File	<code>np.loadtxt('file.txt')</code>
Importing CSV File	<code>np.genfromtxt('file.csv', delimiter=',')</code>
Write Text File	<code>np.savetxt('file.txt', arr, delimiter=' ')</code>

## Data Types

Signed 64-bit integer types	<code>np.int64</code>
Standard double-precision floating point	<code>np.float32</code>
Complex numbers represented by 128 floats	<code>np.complex</code>
Boolean type storing TRUE & FALSE values	<code>np.bool</code>
python object type	<code>np.object</code>
Fixed-length string type	<code>np.string_</code>
Fixed-length unicode type	<code>np.unicode_</code>

## Sorting Array

Sorting 1D Array	<code>arr.sort()</code>
Sorting along the first axis of the 2D array	<code>np.sort(a, axis = 0)</code>

## NumPy Array Manipulation

### Appending Elements to Array

#### One-Dimensional array

Python3  
Adding the values at the end

- of a numpy array  
`print("Original Array:", arr)`
- appending to the array  
`arr = np.append(arr, [7])`  
`print("Array after appending:", arr)`

Output:

```
Original Array: [[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9. 10. 11. 12.]]
Array after appending: [ 1.  2.  3.  4.  5.  6.  7.  8.
 9. 10. 11. 12.  7.]
```

#### N-Dimensional Array

Python3  
Adding the values at the end

- of a numpy array  
`arr = np.arange(1, 13).reshape(2, 6)`  
`print("Original Array")`  
`print(arr, "\n")`
- create another array which is
  - to be appended column-wise  
`col = np.arange(5, 11).reshape(1, 6)`  
`arr_col = np.append(arr, col, axis=0)`  
`print("Array after appending the values column wise")`  
`print(arr_col, "\n")`
  - to be appended row wise  
`row = np.array([1, 2]).reshape(2, 1)`  
`arr_row = np.append(arr, row, axis=1)`  
`print("Array after appending the values row wise")`  
`print(arr_row)`

Output:

```
Original Array
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
Array after appending the values column wise
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [ 5  6  7  8  9 10]]
Array after appending the values row wise
[[ 1  2  3  4  5  6  1]
 [ 7  8  9 10 11 12  2]]
```

### Inserting Elements into the Array

#### One-Dimensional array

Python3  
`arr = np.asarray([1, 2, 3, 4])`

- Python Program illustrating `numpy.insert()`  
`print("1D arr:", arr)`  
`print("Shape:", arr.shape)`
- Inserting value 9 at index 1  
`a = np.insert(arr, 1, 9)`  
`arr.print("\nArray after insertion:", a)`  
`print("Shape:", a.shape)`

## Removing Elements from Numpy Array

### One-Dimensional array

Python3  
Python Program illustrating

- `numpy.delete()`  
`print("Original arr:", arr)`  
`print("Shape : ", arr.shape)`
- deletion from 1D array  
`object = 2`  
`a = np.delete(arr, object)`  
`print("\ndeleting the value at index {} from array:\n {}".format(object,a))`  
`print(arr, "\n")`

## Reshaping Array

### Python3

- creating a numpy array  
`array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])`
- printing array  
`print("Array: " + str(array))`
- reshaping numpy array
  - converting it to 2-D from 1-D array  
`reshaped1 = array.reshape((4, array.size//4))`
  - printing reshaped array  
`print("First Reshaped Array:")`  
`print(reshaped1)`
  - creating another reshaped array  
`reshaped2 = np.reshape(array, (2, 8))`
  - printing reshaped array  
`print("\nSecond Reshaped Array:")`  
`print(reshaped2)`

## Resizing an Array

Numpy arrays can be resized using the `resize()` function. It returns nothing but changes the original array.

### Python3

- Making a random array  
`arr = np.array([1, 2, 3, 4, 5, 6])`
- Required values 12, existing values 6  
`arr.resize(3, 4)`  
`print(arr)`

## Flatten a Two Dimensional array

### One-Dimensional array

#### Python3

- Two dimensional numpy array  
`list_1 = [1, 2, 3, 4]`  
`list_2 = [5, 6, 7, 8]`  
`arr = np.array([list_1, list_2])`  
  
`print(arr.flatten())`

## Transpose

#### Python3

- making a 3x3 array  
`gfg = np.array([[1, 2], [4, 5], [7, 8]])`
- before transpose  
`print(gfg, end = '\n\n')`
- after transpose  
`print(gfg.transpose(1, 0))`

## Combining and Splitting Commands

Combining Arrays	<code>np.concatenate((arr1, arr2), axis = 0)</code>
Splitting array	<code>np.split(arr, 3, 1)</code>
Horizontal Split	<code>np.hsplit(arr, 3)</code>
Vertical Split	<code>np.vsplit(a, 3)</code>

## Indexing, Slicing and Subsetting

### Subsetting Numpy Array

#### Python3

- Index values can be negative.  
`print(arr)`  
`print("Elements are:", arr[np.array([1, 3, -3])])`

### Slicing Numpy Array

The `" "` operator means all elements till the end.

#### Python3

`print(arr)`

- `a[start:stop:step]`  
`print("a[-2:7:1] = ",arr[-2:7:1])`  
`print("a[1:] = ",arr[1:])`

## Indexing Numpy Array

Numpy array indexing is of two types: Integer indexing and Boolean indexing.

#### Python3

- Integer Indexing  
`a = np.array([[1, 2 ],[3, 4 ],[5, 6 ]])`  
`print(a[0, 1, 2 ],[0, 0, 1])`
- Boolean Indexing  
`a = np.array([10, 40, 80, 50, 100])`  
`print(a[a>50])`

## Copying and Viewing Array

Coping to new memory space	<code>arr.copy()</code>
Shallow Copy	<code>arr.view()</code>

## NumPy Array Mathematics

### Arithmetic Operations

Adds elements of 2 Array	<code>np.add(a, b)</code>
Subtracts elements of 2 Array	<code>np.subtract(a, b)</code>
Multiply elements of 2 Array	<code>np.multiply(a, b)</code>
Divide elements of 2 Array	<code>np.divide(a, b)</code>
Modulo elements of 2 Array	<code>np.mod(a, b)</code>
Remainder elements of 2 Array	<code>np.remainder(a,b)</code>
Power elements of 2 Array	<code>np.power(a, b)</code>
Exponent elements of 2 Array	<code>np.exp(b)</code>

### Comparison

#### Python3

- `an_array = np.array([[1, 2], [3, 4]])`  
`another_array = np.array([[1, 2], [3, 4]])`  
`comparison = an_array == another_array`  
`equal_arrays = comparison.all()`  
`print(equal_arrays)`

## Vector Math

### Python3

`arr = np.array([1.5, 1.5, 2.5, 3.5, 4.5, 10.1])`

- `numpy.delete()`  
`print("Original arr:", arr)`  
`print("Shape : ", arr.shape)`
- applying `sqrt()` method  
`print("Square-root:", np.sqrt(arr))`
- applying `log()` method  
`print("Log Value: ", np.log(arr))`
- applying `absolute()` method  
`print("Absolute Value:", np.absolute(arr))`
- applying `sin()` method  
`print("Sine values:", np.sin(arr))`
- applying `ceil()` method  
`print("Ceil values:", np.ceil(arr))`
- applying `floor()` method  
`print("Floor Values:", np.floor(arr))`
- applying `round_()` method  
`print("Rounded values:", np.round_(arr))`

Output:

```
[[1.  1.]
 [1.  1.]]
```

## corrcoef

### Python3

create numpy 1d-array

- create numpy 1d-array  
`array1 = np.array([0, 1, 2])`  
`array2 = np.array([3, 4, 5])`
- pearson product-moment correlation - coefficients of the arrays  
`rs1t = np.corrcoef(array1, array2)`  
  
`print(rs1t)`

Output:

```
[[1.  1.]
 [1.  1.]]
```

## Statistic

### Python3

- `numpy.delete()`  
`print("Original arr:", arr)`  
`print("Shape : ", arr.shape)`
- 1D array  
`arr = [20, 2, 7, 1, 34]`
- mean  
`print("mean of arr:", np.mean(arr))`
- median  
`print("median of arr:", np.median(arr))`
- sum  
`print("Sum of arr(uint8):", np.sum(arr, dtype = np.uint8))`  
`print("Sum of arr(float32):", np.sum(arr, dtype = np.float32))`
- min and max  
`print("maximum element:", np.max(arr))`  
`print("minimum element:", np.min(arr))`
- var  
`print("var of arr:", np.var(arr))`  
`print("var of arr(float32):", np.var(arr, dtype = np.float32))`
- standard deviation  
`print("std of arr:", np.std(arr))`  
`print("More precision with float32", np.std(arr, dtype = np.float32))`