```
import numpy as np
```

```
np.__version__
```

List revisited

```
l_1d = list(range(1, 11))
```

```
l_1d
```

```
l_2d = []
for i in range(1, 11):
    l = list()
    for j in range(1, 4):
        l.append(i**j)

    l_2d.append(l)
```

```
l_2d
```

```
# 2d list with list comprehension
l_2d = [[i**j for j in range(1, 4)] for i in range(1, 11)]
```

```
l = list(range(1, 11))
l.append("football")
l.append("2")
l
```

List vs numpy array

```
arr = np.arange(20)
l = list(range(20))
```

```
arr
```

```
arr = np.arange(200000)
l = list(range(200000))
```

```
import time
```

```
start  = time.time()
for i in range(20):
    for a in arr:
        arr1 = arr**3
end = time.time()
print(end - start)
```

```
start  = time.time()
l = np.arange(20000000)
end = time.time()
print(end - start)
```

```
# Time taken by numpy array and loop
%time for i in range(20): arr1 = arr**3
```

```
%time for i in range(20): l1 = [x**3 for x in l]
```

```
# Creating a numpy array from list
l_1d = [1,2,3,4]
arr_1d = np.array(l_1d)
```

```
arr_1d
```

```
l_2d = [[1,2], [3,4], ["abc", "def"]]
arr_2d = np.array(l_2d, dtype=int)
```

```
arr_2d
```

Some numpy functions

```
arr = np.arange(10)
```

```
arr
```

```
type(arr)
```

```
arr.shape
```

```
arr.ndim
```

```
arr.size
```

```
arr.dtype
```

```
float_arr = arr.astype(np.float64)
float_arr
```

```
arr
```

```
np.isnan(float_arr)
```

```
float_arr[[2, 5, 6]] = np.nan
float_arr
```

```
np.isnan(float_arr)
```

```
# Get all the nan values
float_arr[np.isnan(float_arr)]
```

```
# Get all not nan values
float_arr[~np.isnan(float_arr)]
# float_arr.astype(np.int)
```

```
np.linspace(1, 10, 50)
```

```
# Creating array of ones
# np.ones(2, dtype=int)
np.ones((2,3), dtype = int)
```

```
# Creating array of zeros
np.zeros(5)
```

```
np.zeros((5,6))
```

```
# Diagonal elements 1
np.diag([1,2,3,4])
```

```
np.eye(5, dtype=int)
```

```
# Empty array
# Optional C parameter with C style or Fortran style \
# Default value of float
np.empty(5)
```

```
# Reshaping the shape of array
arr_1d = np.arange(1, 21) # 1d
arr_2d = arr_1d.reshape((5,4))
arr_2d
```

```
arr_2d.ndim
```

Random module and numpy array

```
# Alias for random_sample
# [0.0, 1.0) Half open interval
np.random.random()
```

```
np.random.random(size=5)
```

```
np.random.random(size=(5,3))
```

```
# Closed interval [0,1]
np.random.rand(5)
```

```
np.random.rand(4,3)
```

```
# Standard normal distribution (mean of 0 and std of 1)
np.random.randn(3)
```

```
np.random.randn(4,3,)
```

```
# Number between half open interval
np.random.randint(5, 9)
```

```
np.random.randint(5, 9, size=10)
```

```
# Replace - True allows repetition while false provides unique elements
# Another thing for replace to be false the samples should be less or equal to population
np.random.choice([5, 6, 9, 10])
```

Numpy Arithmetic

```
l = list(range(1, 100))
```

```
l
```

```
l + [2]
```

```
# List cannot be broadcasterd
# List not applicable for numerical computation
l + 2
```

```
# Broadcasting demo
arr = np.arange(1,5)
arr + 2
```

```
arr**3
```

```
arr1 = np.array([[1,3,5], [2,4,6]])
arr2 = np.array([4, 7, 9])
```

```
arr1 + arr2
```

```
arr1 + np.array([4,7])
```

```
arr1 * arr2
```

```
arr2 - arr1
```

```
arr1 / arr2
```

```
arr1 % arr2
```

Indexing and Slicing

```
arr = np.array([11, 19, 23, 29, 37, 65, 89, 102, 234, 589, 876])
```

```
arr
```

```
arr[4]
```

```
arr[4:8]
```

```
arr[::-1]
```

```
arr[::-2]
```

```
arr[:]
```

```
arr[::]
```

```
arr[3:]
```

```
arr[:3]
```

```
arr[::]
```

```
arr
```

```
arr[1:8:2]
```

```
arr
```

```
# Array slices are the views, any change will affect the original array
arr[3:] = 101
```

```
arr
```

```
# Use .copy() if you want the original as it is
arr_copy = arr[:3].copy()
arr_copy = 25
arr
```

```
arr_copy
```

```
arr_2d = np.arange(1, 21).reshape(4,5)
```

```
arr_2d
```

```
arr_2d[1:3, 2:4]
```

```
arr_2d[3]
```

```
arr_2d[:, :3]
```

```
arr_2d[:]
```

Boolean Indexing

```
arr = np.arange(1, 11)
arr == 2
```

```
arr > 2
```

```
arr[arr > 2]
```

```
arr_2d = np.arange(1, 21).reshape(4, 5)
arr_2d < 15
```

```
arr_2d[arr_2d > 15]
```

Dot product

```
a = np.arange(1, 4)
b = np.array([4, -5, 6])
```

```
np.dot(a, b)
```

```
a @ b
```

```
a.dot(b)
```

```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([[1,5], [-1, 6], [4, 9]])
```

```
a @ b
```

Transpose

```
a
```

```
a.T
```

```
a = np.arange(1, 11).T
a
```

Maximum,minimum and sum

```
a.max()
```

```
a.max(axis=1) # Maximum values of columns
```

```
a.max(axis=0) # Maximum values of rows
```

```
a.min(axis=1) # Minimum values of columns
```

```
a.sum(axis=1)
```

```
a.sum(axis=1)
```

```
a.sum(axis=0)
```

```
a.mean()
```

```
a.mean(axis=1)
```

Reducing dimension to one

```
arr = np.arange(1, 21).reshape(4,5)
arr
```

```
# Ravel returns view any change will affect the original array
arr.ravel()
```

```
# Flatten returns a copy, original array won't be affected
arr.flatten()
```

Concatenation

```
a = np.array([[1,3,5],[2,4,6]])
b = np.array([[7,9,11],[8,10,12]])
```

```
np.concatenate([a,b])
```

```
np.concatenate([a,b], axis=1
```

```
np.hstack((a,b))
```

```
np.vstack((a,b))
```