Numpy Array

In [1]:
```
import numpy as np

np.array([5, 9, 13],ndmin=3)
```
Out[1]:
```
array([[[ 5,  9, 13]]])
```

In [ ]:
```
type([5, 9, 13])
```

In [ ]:
```
#Upcasting:
```

In [ ]:
```
np.array([1, 2,9])
```

In [ ]:
```
#two dimensions
```

In [ ]:
```
a = np.array([[1, 2], [3, 4], [4,5]])
a.dtype
```

In [ ]:
```
a
```

In [ ]:
```
#Minimum dimensions 2:
```

In [ ]:
```
#dtype
```

In [ ]:
```
 np.array([1, 2, 3], dtype=float)
```

In [ ]:
```
#Creating array using .mat (matrix)

np.array(np.mat('1 2 3; 3 4 5'))
```

In [ ]:
```
#Convert a list into an array:
```

In [ ]:
```
 a = [1, 2]

np.array(a)
```

In [ ]:
```
b=np.asarray(a)
```

In [ ]:
```
b
```

data types

In [ ]:
```
my_list = [1,2,3]

arr = np.array(my_list)

print("Type/Class of this object:",type(arr))

print("Here is the vector\n--------------------\n",arr)
```

In [ ]:
```
my_mat = [[1,2,3],[4,5,6],[7,8,9]]

mat = np.array(my_mat)

print("Type/Class of this object:",type(mat))

print("Here is the matrix\n----------\n",mat,"\n----------")

print("Dimension of this matrix: ",mat.ndim,sep='') #ndim gives the dimensison, 2 for a matrix, 1 for a vector

print("Size of this matrix: ", mat.size, sep='') #size gives the total number of elements
print("Shape of this matrix: ", mat.shape) #shape gives the number of elements along each axes (dimension)
print("Data type of this matrix: ", mat.dtype) #dtype gives the data type contained in the array

```

In [ ]:
```

```

arange and linspace

In [ ]:
```
list(range(2,10.5,.4))
```

In [ ]:
```
a = np.arange(2,10.5,.4)
a
```

In [ ]:
```
#Reverse order
a[::-1]
```

In [ ]:
```
for i in np.arange(2,10.5,.4):
    if i == 5.6:
        print(i)
    else:
        continue
```

In [ ]:
```
print("Every 5th number from 50 in reverse order\n",np.arange(50,0,-5))
```

In [ ]:
```
print("linearly spaced numbers between 1 and 5\n--------------------------------------------")
print(np.linspace(10,30,4))
```

In [ ]:
```

```

Matrix creation

In [ ]:
```
print("Vector of zeroes\n---------------------")

print(np.zeros(5))
```

In [ ]:
```
print("Matrix of zeroes\n--------------------")

print(np.zeros((3,4))) # Notice Tuples
```

In [ ]:
```
print("Vector of ones\n---------------------")
print(np.ones(10))
```

In [ ]:
```
print("Matrix of ones\n---------------------")
print(np.ones((5,2))) # Note matrix dimension specified by Tuples

```

In [ ]:
```
print("Matrix of 5's\n---------------------")

print(15*np.ones((10,5)))
```

In [ ]:
```
mat1 = np.eye(10)
print("Identity matrix of dimension", mat1.shape)
print(mat1)
```

In [ ]:
```
np.linspace(2.0, 3.0, num=4, endpoint=True)
```

In [ ]:
```
# retstep is the stepsize seperating from samples
np.linspace(2.23456, 10.3587645, num=25, retstep=True)
```

In [ ]:
```
np.logspace(2.0, 3.0, num=4, endpoint=True)
```

In [ ]:
```
#base**start
np.logspace(2.0, 3.0, num=4, base=3.0)
```

In [ ]:
```
#Construct diagonal matrix
x = np.arange(30).reshape(((6,5)))
```

In [ ]:
```
x
```

In [ ]:
```
np.diag(x)
```

In [ ]:
```
np.diag(x, k=-1)
```

In [ ]:
```
np.diag(np.diag(x))
```

In [ ]:
```

```

In [ ]:
```
#Create a two-dimensional array with the flattened input as a diagonal.
```

In [ ]:
```
np.diagflat([[1,2], [3,4],[5,6]])
```

In [ ]:
```
np.diagflat([1,2,3,4], 4)
```

Random number generation

```
In [ ]: print("Random number generation (from Uniform distribution)")
        print(np.random.rand(10,5))
```

```
In [ ]: print("Numbers from Normal distribution with zero mean and standard deviation 1 i.e. standard normal")
        print(np.random.randn(10,2))
```

```
In [ ]: print("Random integer vector:",np.random.randint(1,6,10)) #randint (low, high, # of samples to be drawn)

```

```
In [ ]: print ("\nRandom integer matrix")
        print(np.random.randint(1,30,(4,4))) #randint (low, high, # of samples to be drawn in a tuple to form a matrix)
```

```
In [ ]: print("\n20 samples drawn from a dice throw:",np.random.randint(1,7,20)) # 20 samples drawn from a dice throw
```

Reshaping

```
In [ ]: import numpy as np
        from numpy.random import randint as ri

        a = ri(1,99,30)

        b = a.reshape(2,3,5)

        c = a.reshape(6,5)

```

```
In [ ]: a
```

```
In [ ]: b
```

```
In [ ]: c
```

```
In [ ]: print ("Shape of a:", a.shape)
        print ("Shape of b:", b.shape)
        print ("Shape of c:", c.shape)


```

```
In [ ]: print("\na looks like\n",'-'*20,"\n",a,"\n",'-'*20)
        print("\nb looks like\n",'-'*20,"\n",b,"\n",'-'*20)
        print("\nc looks like\n",'-'*20,"\n",c,"\n",'-'*20)

```

```
In [ ]: A = ri(1,100,10) # Vector of random interegrs

        print("\nVector of random integers\n",'-'*50,"\n",A)

        print("\nHere is the sorted vector\n",'-'*50,"\n",np.sort(A, kind='quicksort'))

```

```
In [ ]: M = ri(1,100,25).reshape(5,5) # Matrix of random interegrs

        print("\n5x5 Matrix of random integers\n",'-'*50,"\n",M)

        print("\nHere is the sorted matrix along each row\n",'-'*50,"\n",np.sort(M)) # Default axis =1

        print("\nHere is the sorted matrix along each column\n",'-'*50,"\n",np.sort(M, axis=0, kind='mergesort'))
```

```
In [ ]: M
```

```
In [ ]: print("\nHere is the sorted matrix along each column\n",'-'*50,"\n",np.sort(M, axis=0, kind='mergesort'))
```

```
In [ ]: a
```

```
In [ ]: b
```

```
In [ ]: print("Max of a:", a.max())
```

```
In [ ]: print("Max of b:", b.max())
```

Indexing and slicing

```
In [ ]: arr = np.arange(0,11)

        print("Array:",arr)

```

In [ ]:
```
print("Element at 7th index is:", arr[7])

```

In [ ]:
```
print("Elements from 3rd to 5th index are:", arr[3:6])

```

In [ ]:
```
print("Elements up to 4th index are:", arr[4:])
abc = arr[4:]
```

In [ ]:
```
arr
```

In [ ]:
```
print("Elements from last backwards are:", arr[-1::-2])

```

In [ ]:
```
print("7 Elements from last backwards are:", arr[-1:-8:-1])
```

In [ ]:
```
arr = np.arange(0,21,2)

print("New array:",arr)

```

In [105]:
```
print("Elements at 2nd, 4th, and 9th index are:", arr[[2,4,9]]) # Pass a list as a index to subset
```

```
Elements at 2nd, 4th, and 9th index are: [ 4  8 18]
```

In [106]:
```
mat
```

Out[106]:
```
array([[11, 12, 13],
       [21, 22, 23],
       [31, 32, 33]])
```

In [107]:
```
print("\nDouble bracket indexing\n------------------------")
print("Element in row index 1 and column index 2:", mat[2][1])

```

```

Double bracket indexing
------------------------
Element in row index 1 and column index 2: 32
```

In [111]:
```
mat
```

Out[111]:
```
array([[11, 12, 13],
       [21, 22, 23],
       [31, 32, 33]])
```

In [112]:
```
print("Entire row at index 2:", mat[2])
print("Entire column at index 3:", mat[:,1:])

```

```
Entire row at index 2: [31 32 33]
Entire column at index 3: [[12 13]
 [22 23]
 [32 33]]
```

In [114]:
```
print("\nSubsetting sub-matrices\n--------------------------")
print("Matrix with row indices 1 and 2 and column indices 3 and 4\n", mat[1:2,1:2])

```

```

Subsetting sub-matrices
--------------------------
Matrix with row indices 1 and 2 and column indices 3 and 4
 [[22]]
```

In [113]:
```
mat
```

Out[113]:
```
array([[11, 12, 13],
       [21, 22, 23],
       [31, 32, 33]])
```

In [115]:
```
print("", mat[0:2,[1,2]])
```

```
 [[12 13]
 [22 23]]
```

Subseting

In [116]:
```
mat = np.array(ri(10,100,15)).reshape(3,5)

print("Matrix of random 2-digit numbers\n--------------------------------\n",mat)

```

```
Matrix of random 2-digit numbers
--------------------------------
 [[35 69 49 30 56]
 [75 78 26 75 18]
 [10 68 25 58 86]]
```

In [118]:
```
print ("Elements greater than 50\n", mat[mat>50])
```

```
Elements greater than 50
 [69 56 75 78 75 68 58 86]
```

In [117]:
```
mat>50
```

Out[117]:
```
array([[False,  True, False, False,  True],
       [ True,  True, False,  True, False],
       [False,  True, False,  True,  True]])
```

Slicing

In [119]:
```
mat = np.array([[11,12,13],[21,22,23],[31,32,33]])

print("Original matrix")

print(mat)

```

```
Original matrix
[[11 12 13]
 [21 22 23]
 [31 32 33]]
```

In [120]:
```
mat_slice = mat[:2,:2]

print ("\nSliced matrix")
print(mat_slice)
print ("\nChange the sliced matrix")
```

```

Sliced matrix
[[11 12]
 [21 22]]

Change the sliced matrix
```

In [122]:
```
mat_slice[0,0] = 1000
print (mat_slice)
```

```
[[1000   12]
 [  21   22]]
```

In [123]:
```
print("\nBut the original matrix? WHOA! It got changed too!")
print(mat)
```

```

But the original matrix? WHOA! It got changed too!
[[1000   12   13]
 [  21   22   23]
 [  31   32   33]]
```

In [124]:
```
# Little different way to create a copy of the slixed matrix
print ("\nDoing it again little differently now...\n")

mat = np.array([[11,12,13],[21,22,23],[31,32,33]])

print("Original matrix")
print(mat)

```

```

Doing it again little differently now...

Original matrix
[[11 12 13]
 [21 22 23]
 [31 32 33]]
```

Universal Functions

In [125]:
```
mat1 = np.array(ri(1,10,9)).reshape(3,3)
mat2 = np.array(ri(1,10,9)).reshape(3,3)

print("\n1st Matrix of random single-digit numbers\n----------------------------------------\n",mat1)
print("\n2nd Matrix of random single-digit numbers\n----------------------------------------\n",mat2)

```

```

1st Matrix of random single-digit numbers
----------------------------------------
 [[8 6 2]
 [4 7 9]
 [4 3 3]]

2nd Matrix of random single-digit numbers
----------------------------------------
 [[6 7 9]
 [3 9 5]
 [7 2 9]]
```

In [126]:
```
print("\nAddition\n------------------\n", mat1+mat2)

print("\nMultiplication\n------------------\n", mat1*mat2)
```

```

Addition
------------------
 [[14 13 11]
 [ 7 16 14]
 [11  5 12]]

Multiplication
------------------
 [[48 42 18]
 [12 63 45]
 [28  6 27]]
```

In [127]:
```
print("\nDivision\n------------------\n", mat1/mat2)
print("\nLineaer combination: 3*A - 2*B\n-----------------------------\n", 3*mat1-2*mat2)
```

```

Division
------------------
 [[1.33333333 0.85714286 0.22222222]
 [1.33333333 0.77777778 1.8       ]
 [0.57142857 1.5        0.33333333]]

Lineaer combination: 3*A - 2*B
-----------------------------
 [[ 12   4 -12]
 [  6   3  17]
 [ -2   5  -9]]
```

In [128]:
```
print("\nAddition of a scalar (100)\n-------------------------\n", 100+mat1)
new_mat = 100+mat1
```

```

Addition of a scalar (100)
-------------------------
 [[108 106 102]
 [104 107 109]
 [104 103 103]]
```

In [129]:
```
new_mat
```

Out[129]:
```
array([[108, 106, 102],
       [104, 107, 109],
       [104, 103, 103]])
```

In [130]:
```
print("\nExponentiation, matrix cubed here\n----------------------------------------\n", mat1**3)
print("\nExponentiation, sq-root using pow function\n-------------------------------------------\n",pow(mat1,0.5))
```

```

Exponentiation, matrix cubed here
----------------------------------------
 [[512 216   8]
 [ 64 343 729]
 [ 64  27  27]]

Exponentiation, sq-root using pow function
-------------------------------------------
 [[2.82842712 2.44948974 1.41421356]
 [2.         2.64575131 3.        ]
 [2.         1.73205081 1.73205081]]
```

In [131]:
```
mat1
```
Out[131]:
```
array([[8, 6, 2],
       [4, 7, 9],
       [4, 3, 3]])
```
In [132]:
```
mat2
```
Out[132]:
```
array([[6, 7, 9],
       [3, 9, 5],
       [7, 2, 9]])
```

Broadcasting

In [ ]:
```
#NumPy operations are usually done on pairs of arrays on an element-by-element basis.
#In the simplest case, the two arrays must have exactly the same shape.
#NumPy’s broadcasting rule relaxes this constraint when the arrays’ shapes meet certain constraints.
#When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing
#dimensions, and works its way forward. Two dimensions are compatible when
#they are equal, or one of them is 1
```
In [133]:
```
start = np.zeros((3,3))
print(start)
```
```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```
In [134]:
```
# create a rank 1 ndarray with 3 values
add_rows = np.array([1, 0, 2])
print(add_rows)
```
```
[1 0 2]
```
In [135]:
```
y = start + add_rows  # add to each row of 'start' using broadcasting
print(y)
```
```
[[1. 0. 2.]
 [1. 0. 2.]
 [1. 0. 2.]]
```
In [136]:
```
# create an ndarray which is 4 x 1 to broadcast across columns
add_cols = np.array([[0,1,2,3]])

add_cols = add_cols.T

print(add_cols)
```
```
[[0]
 [1]
 [2]
 [3]]
```
In [137]:
```
start
```
Out[137]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```
In [146]:
```
# this will just broadcast in both dimensions
add_scalar = np.array([100])

print(start + add_scalar)
```
```
[[100. 100. 100.]
 [100. 100. 100.]
 [100. 100. 100.]]
```

Array Math

In [139]:

```python
mat1 = np.array(ri(1, 10, 9)).reshape(3, 3)
mat2 = np.array(ri(1, 10, 9)).reshape(3, 3)

print("\n1st Matrix of random single-digit numbers\n----------------------------------------\n", mat1)
print("\n2nd Matrix of random single-digit numbers\n----------------------------------------\n", mat2)
```

```

1st Matrix of random single-digit numbers
----------------------------------------
 [[7 6 5]
 [7 3 4]
 [4 8 2]]

2nd Matrix of random single-digit numbers
----------------------------------------
 [[6 9 1]
 [8 3 4]
 [6 4 3]]
```

In [140]:

```python
print("\nSq-root of 1st matrix using np\n------------------\n", np.sqrt(mat1))
```

```

Sq-root of 1st matrix using np
------------------
 [[2.64575131 2.44948974 2.23606798]
 [2.64575131 1.73205081 2.        ]
 [2.         2.82842712 1.41421356]]
```

In [141]:

```python
print("\nExponential power of 1st matrix using np\n", '-'*50, "\n", np.exp(mat1))
```

```

Exponential power of 1st matrix using np
 -------------------------------------------------- 
 [[1096.63315843  403.42879349  148.4131591 ]
 [1096.63315843   20.08553692   54.59815003]
 [  54.59815003 2980.95798704    7.3890561 ]]
```

In [142]:

```python
print("\n10-base logarithm on 1st matrix using np\n",'-'*50,"\n", np.log10(mat1))
```

```

10-base logarithm on 1st matrix using np
 -------------------------------------------------- 
 [[0.84509804 0.77815125 0.69897   ]
 [0.84509804 0.47712125 0.60205999]
 [0.60205999 0.90308999 0.30103   ]]
```

In [143]:

```python
np.log(3)
```

Out[143]: 1.0986122886681098

In [ ]:

```python

```

In [ ]:

```python

```