numPyNotes

December 18, 2022

1 Numpy

1.1 How to import numpy

- if we don't have numpy we can install it with using this command `pip install numpy`

1- with object

`import numpy as np`

in this case we call any function in numpy using

`np.sin(np.deg2rad(30))`

2- with '*'

`from numpy import *`

in this case we call any function directly without object using

`sin(deg2rad(30))`

1.2 Math Operations

1.2.1 Trigonometric Functions

- in python all Trigonomtric function designed to path angles for it in `rad` form

[ ]:
```
from numpy import *

angle = 30

print('wrong sin: {}'.format(round(sin(angle), 2)))       #wrong becuase we
 ↪should pass angle in rad form
print('sin: {}'.format(round(sin(angle*pi / 180), 2)))
print('sin: {}'.format(round(sin(deg2rad(angle)), 2)))
print('-------------')

print('cos: {}'.format(round(cos(deg2rad(angle)), 2)))
print('-------------')

print(f'tan: {round(tan(deg2rad(angle)), 2)}')
print('-------------')
```

```
wrong sin: -0.99
sin: 0.5
sin: 0.5
-------------
cos: 0.87
-------------
tan: 0.58
-------------
```

1.2.2 Rounding

- `round` : rounds up to the nearest Integer which can be above or below or even equal to the actual value.
- `ceil` : rounds up to the nearest Integer which can be equal to or below the actual value.
- `floor` : rounds up to the nearest Integer which can be equal to or above the actual value.

[ ]:
```python
from numpy import *

print(f'round(10.4): {round(10.4)}')
print(f'round(10.5): {round(10.5)}')
print('-------------')
print(f'floor(10.5): {floor(10.5)}')
print(f'floor(10.6): {floor(10.6)}')
print('-------------')
print(f'ceil(10.5): {ceil(10.5)}')
print(f'ceil(10.4): {ceil(10.4)}')
print('-------------')
```

```
round(10.4): 10
round(10.5): 10
-------------
floor(10.5): 10.0
floor(10.6): 10.0
-------------
ceil(10.5): 11.0
ceil(10.4): 11.0
-------------
```

1.2.3 Mod&Power

- `mod` : do the same of function of `%` to get remainder
- `power`: do the smae of function of `**`

[ ]:
```python
from numpy import *

print(f'mod funciton: {mod(10, 3)}')
print(f'% operator: {10 % 3}')
print('-------------')
```

```
print(f'power funciton: {power(10, 2)}')
print(f'** operator: {10**2}')
print('-------------')
```

```
mod funciton: 1
% operator: 1
-------------
power funciton: 100
** operator: 100
-------------
```

1.3 Arrays

1.3.1 1D Arrays

[ ]:
```
from numpy import *

ls = [1, 2, 3, 4, 5]

arr =array(ls)

print(ls)       #note that numbers seperated with ',' that means it's list
print(type(ls))
print('-------------')

print(arr)
print(type(arr))
print('-------------')
```

```
[1, 2, 3, 4, 5]
<class 'list'>
-------------
[1 2 3 4 5]
<class 'numpy.ndarray'>
-------------
```

1.3.2 2D Arrays

[ ]:
```
from numpy import *

ls = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

grid = array(ls)

print(ls)       #note that numbers seperated with ',' that means it's list
print(type(ls))
print('-------------')
```

```
print(grid)
print(type(grid))
print('-------------')
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
<class 'list'>
-------------
[[1 2 3]
 [4 5 6]
 [7 8 9]]
<class 'numpy.ndarray'>
-------------
```

create 2D Array using `list comprehension` with use `range()`

[ ]:
```
from numpy import *

ls = [2, 4, 6]

grid = array([range(i, i+3) for i in ls])

print(grid)
print(type(grid))
print('-------------')
```

```
[[2 3 4]
 [4 5 6]
 [6 7 8]]
<class 'numpy.ndarray'>
-------------
```

1.3.3 Naming data in Array

[ ]:
```
from numpy import *

a = array([('x', 3, 4.2), ('y', 4, 5.3), ('z', 5, 6.3)],
          dtype = [('name', 'U5'), ('number', 'i2'), ('value', 'f4')])
                  #'U5' means string with 5 char
                  # 'i2' means integer with 2 digits
                  #'f4' means float with 4 digits

print(a)
print(type(a))
print('-------------')
```

```
[('x', 3, 4.2) ('y', 4, 5.3) ('z', 5, 6.3)]
<class 'numpy.ndarray'>
-------------
```

1.3.4 Empty Array

[ ]:
```python
a = empty((3, 2))        #create empty 3x2 matrix intialized with 0

print(a)
print(type(a))
print('-------------')
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
<class 'numpy.ndarray'>
-------------
```

1.3.5 Random Array

- we can get random values with using `random.uniform(l, r, num_values)`
- `l`: start, `r`: end, `num_values`: how many numbers we wnat `uniform()` function to generate

[ ]:
```python
from numpy import *

a = random.uniform(1, 10)       #generate just one value
b = random.uniform(1, 10, 20)   #generate 20 values

print(a)
print(type(a))
print('-------------')

print(b)
b.sort()
print('---sorted--------')
print(b)
print(type(b))
print('-------------')
```

```
6.6511880322125805
<class 'float'>
-------------
[8.35525037 8.59086426 5.93283675 7.32542051 7.7444691  4.56659844
 2.3300579  7.18839791 8.33632801 5.43938625 1.67678956 8.95482885
 4.91981831 9.64583426 6.60448032 2.12200438 1.49793073 8.93486746
 6.74385838 4.26980647]
---sorted--------
[1.49793073 1.67678956 2.12200438 2.3300579  4.26980647 4.56659844
 4.91981831 5.43938625 5.93283675 6.60448032 6.74385838 7.18839791
 7.32542051 7.7444691  8.33632801 8.35525037 8.59086426 8.93486746
 8.95482885 9.64583426]
<class 'numpy.ndarray'>
```

----

random.random - this method generate `random` numbers from 0 –> 1

[ ]:
```python
from numpy import *

a = random.random((2, 3))        #generate random 2x3 matrix it's values between␣
 ↪0->1

print(a)

a *= 10                          #mutliply it's values by 10 so it's values will␣
 ↪be between 0->10

print(a)

a += 10                       #just adding 10 to it's values so it's values will␣
 ↪be between 10 --> 20
print(a)
```

```
[[0.86014416 0.44903701 0.36945826]
 [0.44994506 0.4913154  0.75343111]]
[[8.60144162 4.49037006 3.69458255]
 [4.49945063 4.91315402 7.53431112]]
[[18.60144162 14.49037006 13.69458255]
 [14.49945063 14.91315402 17.53431112]]
```

random.normal

is used to get normal distribution of some values

[ ]:
```python
from numpy import *

a = random.normal(0, 1, 10)

print(a)
```

```
[-1.98053867  0.16185355  1.33854811  0.98429316  1.74139616 -0.17478025
 -0.48150579  0.36714585 -0.90277739  1.03221693]
```

random.randint - getting random integer values

[ ]:
```python
from numpy import *

a = random.randint(5)            #return just 1 number

b = random.randint(5, size=7)        #return array of size 7

x = random.randint(5, 20, size=7)       #random values will be 5 >= values < 20
```

```
y = random.randint(5, 20, (3, 3))       #random values in matrix of size 3x3

z = random.randint(5, 20, (2, 3, 3))    #rnadom values in 3d matrix of size␣
 ↪2x3x3

print('-----a------')
print(a)
print(type(a))

print('-----b-------')
print(b)
print(type(b))

print('-----x-------')
print(x)
print(type(x))

print('------y------')
print(y)
print(type(y))

print('------z------')
print(z)
print(type(z))
print('-------------')
```

```
-----a------
3
<class 'int'>
-----b-------
[0 4 0 3 1 1 2]
<class 'numpy.ndarray'>
-----x-------
[ 6 14 12  8 17 18  6]
<class 'numpy.ndarray'>
------y------
[[11 14 18]
 [16  7  6]
 [15  5 18]]
<class 'numpy.ndarray'>
------z------
[[[16 15  7]
  [19 11  9]
  [ 6 15  9]]

 [[ 5  7 15]
  [ 8 17 19]
```

```
  [ 7  8  8]]]
<class 'numpy.ndarray'>
-------------
```

- `reshape()` this method we can use it if we have a list of number and we wnat to reshape it in 2d matrix or another shap but we must ensure that `len(list)` is compatible with new shap

[ ]:
```python
from numpy import *

a = random.randint(1, 60, 25)           #generate an array of 25 random␣
 ↪elements in range 1<= vals < 60

b = reshape(a, (5, 5))              #reshape array a to 2D Array of size 5x5

print('-----a-------')
print(a)
print(type(a))

print('-----b------')
print(b)
print(type(b))
print('-------------')
```

```
-----a-------
[35 40 33 46  3 32 40  6 19 15  9 27 58  5  7 46 11 58 22 56 19 52  1  9
 50]
<class 'numpy.ndarray'>
-----b------
[[35 40 33 46  3]
 [32 40  6 19 15]
 [ 9 27 58  5  7]
 [46 11 58 22 56]
 [19 52  1  9 50]]
<class 'numpy.ndarray'>
-------------
```

random.rand() - this mehtod do the same thing of `uniform()` reutrn random values from 0 –> 1 - it has some advantage that if i pass 1 dimension or more than one

[ ]:
```python
from numpy import *

a = random.rand(1)                        #return just array of size 1

b = random.rand(15)                       #return array of size 15

c = random.rand(3, 5)                   #return matrix of size 3x5

d = random.rand(2, 3, 5)            #return 3D matrix of size 2x3x5
```

```
print('------a-------')
print(a)
print(type(a))

print('------b-------')
print(b)
print(type(b))

print('------c------')
print(c)
print(type(c))

print('------d-------')
print(d)
print(type(d))
print('-------------')
```

```
------a-------
[0.52966046]
<class 'numpy.ndarray'>
------b-------
[0.64226043 0.50581055 0.81899322 0.28918178 0.48022607 0.35772815
 0.0188346  0.86704258 0.35917433 0.30371867 0.74986761 0.22731163
 0.76170547 0.70368947 0.90225148]
<class 'numpy.ndarray'>
------c------
[[0.08197236 0.76812523 0.43438084 0.03480177 0.81150968]
 [0.09193523 0.86110363 0.56589955 0.83548073 0.0261031 ]
 [0.90306937 0.25701887 0.05077782 0.61034724 0.33202945]]
<class 'numpy.ndarray'>
------d-------
[[[0.61425652 0.74410471 0.68037153 0.26284637 0.63575013]
  [0.52925445 0.26927961 0.8675419  0.90639509 0.13791938]
  [0.88010914 0.88717705 0.69461119 0.02455665 0.44455293]]

 [[0.11740865 0.73864846 0.9707813  0.79901074 0.70954128]
  [0.17980517 0.32790773 0.09528888 0.76472996 0.86061068]
  [0.04882949 0.80872788 0.08407256 0.03933752 0.15818748]]]
<class 'numpy.ndarray'>
-------------
```

random.choice() - return random value from list

[ ]:
```
from numpy import *

y = [1, 2, 3, 5, 6, 10]
a = random.choice(y)
```

```
print(a)
print(type(a))
print('-------------')
```

```
3
<class 'numpy.int32'>
-------------
```

random.shuffle() - using to randomly reorder `list` or `array`

[ ]:
```
from numpy import *

y = [1, 2, 3, 6, 9, 8, 5, 4, 7, 8, 9, 6, 5, 9, 6]

print('-------y before shuffle-----')
print(y)

random.shuffle(y)
print('-------y after shuffle------')
print(y)
print('-------------')
```

```
-------y before shuffle-----
[1, 2, 3, 6, 9, 8, 5, 4, 7, 8, 9, 6, 5, 9, 6]
-------y after shuffle------
[5, 7, 6, 6, 5, 6, 1, 8, 9, 2, 8, 9, 3, 4, 9]
-------------
```

1.3.6 Zeroes & Ones

- `zeroes`: create array of zero initial values
- `ones` : create array of ones initial values

[ ]:
```
from numpy import *

a = zeros(5)

b = ones(10)

print('------a-------')
print(a)
print(type(a))

print('------b-------')
print(b)
print(type(b))
print('-------------')
```

```
------a-------
[0. 0. 0. 0. 0.]
<class 'numpy.ndarray'>
------b-------
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
<class 'numpy.ndarray'>
-------------
```

[ ]:
```
from numpy import *

a = zeros((2, 3))

b = ones((2, 3))

print('------a-------')
print(a)
print(type(a))

print('------b-------')
print(b)
print(type(b))
print('-------------')
```

```
------a-------
[[0. 0. 0.]
 [0. 0. 0.]]
<class 'numpy.ndarray'>
------b-------
[[1. 1. 1.]
 [1. 1. 1.]]
<class 'numpy.ndarray'>
-------------
```

[ ]:
```
from numpy import *

a = zeros((2, 3, 2))

b = ones((2, 3, 2))

print('------a-------')
print(a)
print(type(a))

print('------b-------')
print(b)
print(type(b))
print('-------------')
```

```
------a-------
[[[0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]]]
<class 'numpy.ndarray'>
------b-------
[[[1. 1.]
  [1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]
  [1. 1.]]]
<class 'numpy.ndarray'>
-------------
```

1.3.7 eye

- create identity matrix which is matrix all elements are zeroes and only main diagonal is ones

[ ]:
```
from numpy import *

a = eye(3)

print(a)
print(type(a))
print('--------------')
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
<class 'numpy.ndarray'>
-------------
```

1.3.8 full

- use to fill matrix with specific value

[ ]:
```
from numpy import *
a = full((3, 5), 35)

print(a)
print('--------------')
```

```
[[35 35 35 35 35]
```

```
 [35 35 35 35 35]
 [35 35 35 35 35]]
-------------
```

1.3.9 arange()

- use to get an array contains numbers in range

[ ]:
```python
from numpy import *

a = arange(10)          #return array of element in range 0 <=ele< 10

print(a)
print('-------------')
```

```
[0 1 2 3 4 5 6 7 8 9]
-------------
```

[ ]:
```python
from numpy import *

a = arange(1, 19).reshape(3, 6)

print(a)
print('-------------')
```

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]]
-------------
```

1.3.10 linspace

- returns number spaces with in defined interval similar to `arange()` but `arrange()` use fixed step with 1
- we can convert returned array to matrix with using `reshape()`

[ ]:
```python
from numpy import *

a = linspace(0, 100, 5)

print(a)
print('-------------')
```

```
[  0.  25.  50.  75. 100.]
-------------
```

1.3.11 Diagnoal Matrix

- matrix that main digonal has values and all other elements are Zeros

```
[ ]: from numpy import *

a = diag(array([5, 12, 4, -1, 3]))            #create matrix 5x5 this values␣
 ↪will be on main digonal
b = diag(array([5, 12, 4, -1, 3]), k=3)        #create diagonal matrix of 5x5␣
 ↪with this values then add 3 columns & rows so it will be 8x8

print('------a-------')
print(a)

print('------b-------')
print(b)
print('-------------')
```

```
------a-------
[[ 5  0  0  0  0]
 [ 0 12  0  0  0]
 [ 0  0  4  0  0]
 [ 0  0  0 -1  0]
 [ 0  0  0  0  3]]
------b-------
[[ 0  0  0  5  0  0  0  0]
 [ 0  0  0  0 12  0  0  0]
 [ 0  0  0  0  0  4  0  0]
 [ 0  0  0  0  0  0 -1  0]
 [ 0  0  0  0  0  0  0  3]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]]
-------------
```

1.3.12 Dealing with Matrices

count_nonzero(matrix_name) - this method count numbers in matrix that is not zero and also we can path to it condation like a>5 (a is matrix) so in this case will count matrix elements that greater than 5

```
[ ]: from numpy import *


a = random.randint(0, 10, (3, 3))

print(a)
print(count_nonzero(a))
print(count_nonzero(a > 5))
print(count_nonzero(a < 1))
```

```
[[9 0 7]
```

```
 [2 7 8]
 [0 0 5]]
6
4
3
```

[ ]:
```
from numpy import *

a = random.randint(0, 10, (3, 3))

print(a)
print('-------------')
print(count_nonzero(a < 3, axis=0))                  #count number of non_zeroes␣
 ↪in each column
print(count_nonzero(a < 3, axis=1))                  #count number of non_zeroes␣
 ↪in each row
```

```
[[8 0 6]
 [9 3 6]
 [7 9 4]]
-------------
[0 1 0]
[1 0 0]
```

any(condation) - return true if any element on the matrix satisfye the condation

[ ]:
```
from numpy import *

a = random.randint(0, 10, (3, 3))

print(a)
print('-------------')
print(any(a<5))                                 #return True if any␣
 ↪element in matrix is less than 5
print(any(a<5, axis=0))                         #return list of True or␣
 ↪False for each column
print(any(a<5, axis=1))                         #return list of True or␣
 ↪False for each row
```

```
[[2 8 0]
 [8 5 0]
 [2 8 3]]
-------------
True
[ True False  True]
[ True  True  True]
```

all(condation) - reutrn True if all elements in matrix satisfye the condation

```
[ ]: from numpy import *

     a = random.randint(0, 10, (3, 3))

     print(a)
     print('-------------')

     print(all(a<3))
     print(all(a<3, axis=0))
     print(all(a<3, axis=1))
```

```
[[0 0 0]
 [2 4 3]
 [3 4 6]]
-------------
False
[False False False]
[ True False False]
```

Filter matrix depends on condition

```
[ ]: from numpy import *

     a = random.randint(5, 20, size=9).reshape(3, 3)

     b = a>10                  #matrix of True or False that is the answer of the␣
      ↪condation
     c = a[b]                  #filter a with the condation
     d = a[a<10]

     print(a)
     print('-------------')
     print(b)
     print('-------------')
     print(c)
     print('-------------')
     print(d)
     print('-------------')
```

```
[[17 11 13]
 [11  8  5]
 [14 16 13]]
-------------
[[ True  True  True]
 [ True False False]
 [ True  True  True]]
-------------
[17 11 13 11 14 16 13]
```

```
-------------
[8 5]
-------------
```

Compare Matrices - using method `isclose(matrix1, matrix2, rtol=tolerance)` and return matrix of `True` or `False`

[ ]:
```
from numpy import *

a = arange(9).reshape(3, 3)
b = arange(9).reshape(3, 3)

c = 2*b

print(isclose(a, b, rtol=0.1))
print(isclose(a, c, rtol=0.1))
```

```
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]]
[[ True False False]
 [False False False]
 [False False False]]
```

Multiply constant - we can use `muliply(matrix, constant, out=outputMatrix)` or we can use directly `*` operator

[ ]:
```
from numpy import *

a = arange(5)
b = empty(5)

multiply(a, 10, out=b)

c = a * 10
print(b)
print(c)
```

```
[ 0. 10. 20. 30. 40.]
[ 0 10 20 30 40]
```

power - we can use `power(matrix, exponent, out=outputMatrix)` or we can use directly `**`operator

[ ]:
```
from numpy import *

a = arange(5)
b = empty(5)

power(a, 4, out=b)
```

```
c = a**4

print(b)
print(c)
```

```
[  0.   1.  16.  81. 256.]
[  0   1  16  81 256]
```

reduce - it means array dimension by one, by applaying another function like `add` or `multiply` or other functions

[ ]:
```
from numpy import *

a = arange(9)

print(add.reduce(a))                          #return just one value it's␣
  ↪the sum of all elements in array
print(multiply.reduce(a))                     #return just one value it's␣
  ↪the multiplication answer of all elements in array
print('-------------')

a = a.reshape(3, 3)                           #now a is matrix of 3x3
print(a)
print('-------------')
print(add.reduce(a))                          #return array of summation␣
  ↪of each column
print(multiply.reduce(a))                     #return array of␣
  ↪multiplication of each column
print('-------------')
```

```
36
0
-------------
[[0 1 2]
 [3 4 5]
 [6 7 8]]
-------------
[ 9 12 15]
[ 0 28 80]
-------------
```

Accumulate - Accumulate the result of applying the operator to all elements

[ ]:
```
from numpy import *

a = arange(9)
```

```
print(add.accumulate(a))
print(multiply.accumulate(a))

a = a.reshape(3, 3)
print(add.accumulate(a))
print(multiply.accumulate(a))
```

```
[ 0  1  3  6 10 15 21 28 36]
[0 0 0 0 0 0 0 0 0]
[[ 0  1  2]
 [ 3  5  7]
 [ 9 12 15]]
[[ 0  1  2]
 [ 0  4 10]
 [ 0 28 80]]
```

Outer - Compute the outer product of two vectors.

Given two vectors, a = [a0, a1, ..., aM] and b = [b0, b1, ..., bN], the outer product [1] is:

```
[[a0*b0  a0*b1 ... a0*bN ]
[a1*b0    .
[ ...          .
[aM*b0            aM*bN ]]
```

[ ]:
```
from numpy import *

a = arange(1, 5)

print(add.outer(a, a))
print('-------------')
print(multiply.outer(a, a))
print('-------------')
```

```
[[2 3 4 5]
 [3 4 5 6]
 [4 5 6 7]
 [5 6 7 8]]
-------------
[[ 1  2  3  4]
 [ 2  4  6  8]
 [ 3  6  9 12]
 [ 4  8 12 16]]
-------------
```

Size & Dimensions

- To know know size or dimensions we can use:
  - `len()`: to know size of 1D Array or first dimension of multidimension array
  - `size()`: get number of elements in array (don't care about dimensions)

- `shape()`: return tuple of dimensions
- `ndim()`: return number of dimensions

[ ]:
```python
from numpy import *

a = arange(9)

print(f'len: {len(a)}')
print(f'size: {a.size}')
print(f'ndim: {a.ndim}')
print(f'shape: {a.shape}')
print('-------------')

a = a.reshape(3, 3)
print(f'len: {len(a)}')
print(f'size: {a.size}')
print(f'ndim: {a.ndim}')
print(f'shape: {a.shape}')
print('-------------')
```

```
len: 9
size: 9
ndim: 1
shape: (9,)
-------------
len: 3
size: 9
ndim: 2
shape: (3, 3)
-------------
```

dtype - we can use `dtype` to return type of element in the array

[ ]:
```python
from numpy import *

a = array([1, 2, 3, 4])
b = array([1.2, 3, 3.5])
c = array(['a', 'cd'])

print(a.dtype)
print(b.dtype)
print(c.dtype)
```

```
int32
float64
<U2
```

format - do the same thing of reshape

[ ]:
```
from numpy import *

a = matrix('{} {} ; {} {}'.format(1, 2, 3, 4))
b = matrix('{} {} {}; {} {} {}'.format(1, 2, 3, 4, 5, 6))

print(a)
print('------------------')
print(b)
print('------------------')
```

```
[[1 2]
 [3 4]]
------------------
[[1 2 3]
 [4 5 6]]
------------------
```

trace - to get summation of main diagonal

[ ]:
```
from numpy import *

a = arange(9)
b = a.reshape(3, 3)
c = trace(b)

print(a)
print('------------------')
print(b)
print('------------------')
print(c)
print('------------------')
```

```
[0 1 2 3 4 5 6 7 8]
------------------
[[0 1 2]
 [3 4 5]
 [6 7 8]]
------------------
12
------------------
```

det, eig - `linalg.det()`: to get determinant of matrix - `linalg.eig()`: to get eigen values of matrix

eigen values calculated from this equeation $|A - \lambda I| = 0$

[ ]:
```
from numpy import *

a = arange(9)
```

```
b = a.reshape(3, 3)
c = linalg.det(b)
d = linalg.eig(b)

print(a)
print('------------------')
print(b)
print('------------------')
print(c)
print('------------------')
print(d)
print('------------------')
```

```
[0 1 2 3 4 5 6 7 8]
------------------
[[0 1 2]
 [3 4 5]
 [6 7 8]]
------------------
0.0
------------------
(array([ 1.33484692e+01, -1.34846923e+00, -2.48477279e-16]), array([[
0.16476382,  0.79969966,  0.40824829],
       [ 0.50577448,  0.10420579, -0.81649658],
       [ 0.84678513, -0.59128809,  0.40824829]]))
------------------
```

slicing - the same as in `strings` and `lists`

[ ]:
```
from numpy import *

a = arange(10)

print(a)
print(a[3])
print(a[3:9])
print(a[3:9:2])
print(a[-1])
print(a[-3])
```

```
[0 1 2 3 4 5 6 7 8 9]
3
[3 4 5 6 7 8]
[3 5 7]
9
7
```

[ ]:
```python
from numpy import *

a = arange(36).reshape(6, 6)

print(a)            #printing a
print('------------------')
print(a[3])     #printing 4th row
print('------------------')
print(a[1:3])   #printing 2nd & 3rd row
print('------------------')
print(a[3:9:2])     #printing rows 4, 6, 8 but matrix has 6 rows only
print('------------------')
print(a[-1])            #printing last row
print('------------------')
print(a[1:3, :])            #printing element in 2nd, 3rd row
print('------------------')
print(a[1:3, 1:3])          #printing elements in 2nd, 3rd row and columns 2nd,␣
 ↪3rd
print('------------------')
print(a[1:3, 1:])
print('------------------')
print(a[1:3, 3:])
print('------------------')
print(a[-1, :3])
print('------------------')
print(a[-1, ::3])
print('------------------')
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
------------------
[18 19 20 21 22 23]
------------------
[[ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
------------------
[[18 19 20 21 22 23]
 [30 31 32 33 34 35]]
------------------
[30 31 32 33 34 35]
------------------
[[ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
```

```
------------------
[[ 7  8]
 [13 14]]
------------------
[[ 7  8  9 10 11]
 [13 14 15 16 17]]
------------------
[[ 9 10 11]
 [15 16 17]]
------------------
[30 31 32]
------------------
[30 33]
------------------
```

[ ]:
```
from numpy import *

a = arange(36).reshape(6, 6)

print(a)
print('------------------')
print(a[::2, ::3])
print('------------------')
print(a[::-1, ::-1])
print('------------------')
print(a[:2:-1, :3:-1])
print('------------------')
print(a[2::2, 3::3])
print('------------------')
print(a[-1::, -1::])
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
------------------
[[ 0  3]
 [12 15]
 [24 27]]
------------------
[[35 34 33 32 31 30]
 [29 28 27 26 25 24]
 [23 22 21 20 19 18]
 [17 16 15 14 13 12]
 [11 10  9  8  7  6]
 [ 5  4  3  2  1  0]]
```

```
------------------
[[35 34]
 [29 28]
 [23 22]]
------------------
[[15]
 [27]]
------------------
[[35]]
```

Note:

```
a = arrange(16).reshape(4, 4)
b = a[:, 1:3]          #in this case b is a part of a so if we change a also b will be affect
c = a[:, 1:3].copy()       # in this case c is indepentet copy so if we change a , c won't cha
```

split

```
[ ]: from numpy import *

x = arange(1,9) * 11

print(x)
print('------------------')

x1, x2, x3 = split(x, (3, 6))
print(f'{x1}\t{x2}\t{x3}')
print('------------------')


x1, x2, x3 = split(x, (1, 5))
print(f'{x1}\t{x2}\t{x3}')
print('------------------')

x1, x2, x3 = split(x, (6, 3))
print(f'{x1}\n{x2}\n{x3}')
print('------------------')

x1, x2, x3 = split(x, (0, 3))
print(f'{x1}\t{x2}\t{x3}')
print('------------------')


x1, x2, x3 = split(x, (4, 0))
print(f'{x1}\t{x2}\t{x3}')
print('------------------')
```

```
[11 22 33 44 55 66 77 88]
------------------
```

```
[11 22 33]      [44 55 66]      [77 88]
------------------
[11]     [22 33 44 55]    [66 77 88]
------------------
[11 22 33 44 55 66]
[]
[44 55 66 77 88]
------------------
[]       [11 22 33]       [44 55 66 77 88]
------------------
[11 22 33 44]    []       [11 22 33 44 55 66 77 88]
------------------
```

get Element

[ ]:
```
from numpy import *

a  =arange(9).reshape(3, 3)

print(f'{a[2][1]}')
print(f'{a[2, 1]}')
```

```
7
7
```

merge - we can use - `vstack()`: to merge 2 matrices vertically but must have the same number of columns - `hstack()`: to merge 2 matrices horizontally but must have the same number of rows

[ ]:
```
from numpy import *

a = arange(4).reshape(2, 2)
b = arange(6).reshape(3, 2)
c = b.reshape(2, 3)

v = vstack((a, b))
h = hstack((a, c))

print(v)
print('------------------')
print(h)
print('------------------')
```

```
[[0 1]
 [2 3]
 [0 1]
 [2 3]
 [4 5]]
------------------
[[0 1 0 1 2]
```

```
 [2 3 3 4 5]]
------------------
```

concatenate - when we set `axis=0` doing the same function of `vstack()` - when we set `axis=1` doing the same function of `hstack()`

[ ]:
```
a = random.randint(5, 20, size=9).reshape(3, 3)
b = random.randint(5, 20, size=6).reshape(2, 3)
c = b.reshape(3, 2)

print(a)
print('------------------')
print(concatenate([a, b], axis=0))
print('------------------')
print(concatenate([a, c], axis=1))
print('------------------')
```

```
[[ 6 19 11]
 [14 17 13]
 [ 6 12  5]]
------------------
[[ 6 19 11]
 [14 17 13]
 [ 6 12  5]
 [ 8 10  7]
 [12  6  5]]
------------------
[[ 6 19 11  8 10]
 [14 17 13  7 12]
 [ 6 12  5  6  5]]
------------------
```

max & min - `max()`: get max value in matrix - `min()`: get min value in matrix - `argmax()`: get position of max in matrix - `argmin()`: get position of min in matrix

[ ]:
```
import numpy as np

a = np.random.randint(5, 20, size=9).reshape(3, 3)


print(f'max: {np.max(a)}')
print('------------------')
print(f'min: {np.min(a)}')
print('------------------')
print(f'max pos: {np.argmax(a)}')
print('------------------')
print(f'min pos: {np.argmin(a)}')
print('------------------')
```

```
max: 17
------------------
min: 5
------------------
max pos: 1
------------------
min pos: 7
------------------
```

Variance & Covariance

[ ]:
```python
from numpy import *

a = random.randint(5, 20, size=9).reshape(3, 3)

b = var(a)
c = cov(a)

print(a)
print('------------------')
print(b)
print('------------------')
print(c)
print('------------------')
```

```
[[15  8 14]
 [18  5  5]
 [ 9  9  6]]
------------------
19.65432098765432
------------------
[[14.33333333 17.33333333 -2.5       ]
 [17.33333333 56.33333333  6.5       ]
 [-2.5         6.5         3.        ]]
------------------
```

Mathematical Operations on Matrices

[ ]:
```python
from numpy import *

a = random.randint(5, 20, size= 9).reshape(3, 3)
b = random.randint(5, 20, size=9).reshape(3, 3)

print(a)
print('--------------------')
print(b)
print('--------------------')
print(a+b)
print('--------------------')
```

```
print(a-b)
print('--------------------')
print(a*b)
print('--------------------')
print(a**2)
print('--------------------')
print(log(a))
print('--------------------')
print(dot(a, b))            #product of 2 matrices
print('--------------------')
```

```
[[12 19 16]
 [17 11  9]
 [19 17 16]]
--------------------
[[18 12  8]
 [ 7 12 18]
 [ 6 17 11]]
--------------------
[[30 31 24]
 [24 23 27]
 [25 34 27]]
--------------------
[[-6  7  8]
 [10 -1 -9]
 [13  0  5]]
--------------------
[[216 228 128]
 [119 132 162]
 [114 289 176]]
--------------------
[[144 361 256]
 [289 121  81]
 [361 289 256]]
--------------------
[[2.48490665 2.94443898 2.77258872]
 [2.83321334 2.39789527 2.19722458]
 [2.94443898 2.83321334 2.77258872]]
--------------------
[[445 644 614]
 [437 489 433]
 [557 704 634]]
--------------------
```

[ ]:
```
from numpy import *

a = random.randint(5, 20, size=9).reshape(3, 3)
```

```
b = sum(a)

print(sum(a))
print('---------------------')
print(a.sum(axis=1))
print('---------------------')
print(a.sum(axis=0))
print('---------------------')
```

```
114
--------------------
[22 50 42]
--------------------
[33 37 44]
--------------------
```

mean & standard deviation & variance & correlation coefficient

[ ]:
```
from numpy import *

a = random.randint(5, 20, size=9).reshape(3, 3)

print(a)
print('---------------------')
print(a.mean())
print('---------------------')
print(a.std())
print('---------------------')
print(a.var())
print('---------------------')
print(corrcoef(a))
print('---------------------')
```

```
[[14 17 12]
 [ 8 17 12]
 [11 17  6]]
--------------------
12.666666666666666
--------------------
3.7712361663282534
--------------------
14.222222222222221
--------------------
[[1.         0.64622084 0.9980461 ]
 [0.64622084 1.         0.59727508]
 [0.9980461  0.59727508 1.        ]]
--------------------
```

sorting matrix - we can sort matrix or sort rows or columns - `sort(a, axis=0)`: means sort each columns in matrix a - `sort(a, axis=1)`: means sort each row in matrix a

[ ]:
```python
from numpy import *

a = random.randint(5, 20, size=9).reshape(3, 3)
b = sort(a, axis=0)
c = sort(a, axis=1)
d = sort(a)

print(a)
print('--------------------')
print(b)
print('--------------------')
print(c)
print('--------------------')
print(d)
print('--------------------')
```

```
[[ 8 14 14]
 [10 17  6]
 [ 5 14  9]]
--------------------
[[ 5 14  6]
 [ 8 14  9]
 [10 17 14]]
--------------------
[[ 8 14 14]
 [ 6 10 17]
 [ 5  9 14]]
--------------------
[[ 8 14 14]
 [ 6 10 17]
 [ 5  9 14]]
--------------------
```

Inverse matrix

[ ]:
```python
from numpy import *

a = random.randint(1, 4, size=9).reshape(3, 3)
b = linalg.inv(a)
c = dot(a, b)

print(a)
print('--------------------')
print(b)
print('--------------------')
print(c)
```

```
print('--------------------')
```

```
[[3 1 2]
 [1 2 1]
 [1 3 1]]
--------------------
[[ 1.00000000e+00 -5.00000000e+00  3.00000000e+00]
 [ 2.08166817e-17 -1.00000000e+00  1.00000000e+00]
 [-1.00000000e+00  8.00000000e+00 -5.00000000e+00]]
--------------------
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
--------------------
```

1.4 Files

- we can use numpy to deal with files with using

`loadtxt(fname, dtype, skiprows, usecols)`

- `fname`: file path
- `dtype`: define type of each column
- `skiprows`: each file has headers or paragarph that tell us what is the content of the file so we can skip them with this
- `usecols` : columns we want to load it's data

[ ]:
```
from numpy import *

fname = '..\\jupyterNotes\\txtFiles\\student_data.txt'

dtype1 = dtype([('gender', 'U1'), ('height', 'f8')])

a = loadtxt(fname, dtype=dtype1, skiprows=9, usecols=(1, 3))

print(a)
```

```
[('M', 1.82) ('M', 1.77) ('F', 1.68) ('M', 1.72) ('F', 1.78) ('F', 1.6 )
 ('M', 1.72) ('M', 1.83) ('F', 1.56) ('F', 1.64) ('M', 1.63) ('M', 1.67)
 ('M', 1.66) ('F', 1.59) ('F', 1.7 ) ('M', 1.97) ('F', 1.66) ('F', 1.63)
 ('M', 1.69)]
```

[ ]:
```
from numpy import *

fname = '..\\jupyterNotes\\txtFiles\\marriage_age.txt'

a, b, c = loadtxt(fname, unpack=True, skiprows=3)
print(a)
print(b)
```

```
print(c)
```

```
[1890. 1900. 1910. 1920. 1930.]
[26.1 25.9 25.1 24.6 24.3]
[22.  21.9 21.6 21.2 21.3]
```

[ ]:
```
from numpy import *

fname = '..\\jupyterNotes\\txtFiles\\subject.txt'

data = genfromtxt(fname, skip_header=1,
                  dtype= [('student', 'u8'),
                          ('gender', 'S1'),
                          ('black', 'f8'),
                          ('color', 'f8')], delimiter=',',
                  missing_values='X')

print(data)
```

```
[(1, b'F', 18.72, 31.11) (2, b'F', 21.14, 52.47) (3, b'F', 19.38, 33.9 )]
```

1.5 Ploynomials

1.5.1 fitting

`np.polynomial.Polynomial.fit(x, y, order, full=True)`

- `x` : x values
- `y` : y values
- `order` : order we want to fit to it 1st or 2nd or …
- `full` : True means take all values in x & y

[ ]:
```
import numpy as np

x = np.array([0, 20, 40, 60, 80, 100, 120, 140, 160, 180])
y = np.array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])

points, stats = np.polynomial.Polynomial.fit(x, y, 1, full=True)

print(points)
# print(stats)
```

```
5.500000000000001 - 4.500000000000003 x**1
```

Passing Polynomials - `poly1d((coefficients))`

[ ]:
```
import numpy as np

a = np.poly1d((-7))
b = np.poly1d((-7, 2))
```

```python
c = np.poly1d((-7, 2, 1))
d = np.poly1d((-7, 2, 1, 3))
e = np.poly1d((-7, 2, 1, 3, 6))

print(a)
print('---------------------')
print(b)
print('---------------------')
print(c)
print('---------------------')
print(d)
print('---------------------')
print(e)
print('---------------------')
```

```
-7
---------------------

-7 x + 2
---------------------
    2
-7 x + 2 x + 1
---------------------
    3     2
-7 x + 2 x + 1 x + 3
---------------------
    4     3     2
-7 x + 2 x + 1 x + 3 x + 6
---------------------
```

[ ]:
```python
import numpy as np

a = np.poly1d((-7, 2, 1, 3, 6))

print(a)
print('---------------------')
print(a(-15))
print('---------------------')
print(a(0))
print('---------------------')
```

```
    4     3     2
-7 x + 2 x + 1 x + 3 x + 6
---------------------
-360939
---------------------
6
```

```
---------------------
```

`polyval((polynomial coefficient), val)` : get value of this polynomial at `x = val`

[ ]:
```python
import numpy as np

a = np.polyval((1, 2), 2)
b = np.polyval((1, 2, 3), 7)
c = np.polyval((1, 2, 3, 5), -3)
d = np.polyval((1, 2, 3, 5, -6), 12.6)

print(a)
print('---------------------')
print(b)
print('---------------------')
print(c)
print('---------------------')
print(d)
print('---------------------')
```

```
4
---------------------
66
---------------------
-13
---------------------
29738.769599999992
---------------------
```

Derivative

`polyder(polynomial equation)`

[ ]:
```python
import numpy as np

a_eq = np.poly1d((1, 2, 3))
a_der1 = np.polyder(a_eq, 1)
a_der2 = np.polyder(a_eq, 2)

print(a_eq)
print('---------------------')
print(a_der1)
print('---------------------')
print(a_der2)
print('---------------------')
print(a_der1(2))
print('---------------------')
```

```
   2
1 x + 2 x + 3
```

```
---------------------

2 x + 2
---------------------

2
---------------------
6
---------------------
```

Integration

polyint(polynomial equation)

[ ]:
```python
import numpy as np

a_eq = np.poly1d((1, 2, 3))
a_int = np.polyint(a_eq, 1)

print(a_eq)
print('---------------------')
print(a_int)
print('---------------------')
```

```
   2
1 x + 2 x + 3
---------------------
        3     2
0.3333 x + 1 x + 3 x
---------------------
```

Roots

roots(polynomial equation)

[ ]:
```python
import numpy as np

a_eq = np.poly1d((1, 2, 3))
a_roots = np.roots(a_eq)

b_eq = np.poly1d((1, 2))
b_roots = np.roots(b_eq)

print(a_eq)
print('---------------------')
print(a_roots)
print('---------------------')
print(b_eq)
print('---------------------')
print(b_roots)
```

```
print('---------------------')
```

```
   2
1 x + 2 x + 3
---------------------
[-1.+1.41421356j -1.-1.41421356j]
---------------------

1 x + 2
---------------------
[-2.]
---------------------
```

polyfit

`polyfit(x, y, order)` : also using for fitting

[ ]:
```
import numpy as np

x = np.array([3, 6, 2, 5, 4])
y = np.array([2, 3, -9, 6, 2.5])
z = np.polyfit(x, y, 2)

print(x)
print('---------------------')
print(y)
print('---------------------')
print(z)
print('---------------------')
```

```
[3 6 2 5 4]
---------------------
[ 2.   3.  -9.   6.   2.5]
---------------------
[ -1.78571429  17.08571429 -35.3        ]
---------------------
```

1.6 Data

[ ]:
```
import numpy as np

x = np.array('2015-07-04', dtype=np.datetime64)

y = np.datetime64('2015-07-04')

print(x)
print('---------------------')
print(y)
print('---------------------')
```

```
2015-07-04
---------------------
2015-07-04
---------------------
```

[ ]:
```
import numpy as np

x = np.datetime64('2015-07-04')

y = x + arange(12)

z = x - arange(12)

print(x)
print('---------------------')
print(y)
print('---------------------')
print(z)
print('---------------------')
print(np.datetime64('2022-12-14') - np.datetime64('2022-05-01'))
```

```
2015-07-04
---------------------
['2015-07-04' '2015-07-05' '2015-07-06' '2015-07-07' '2015-07-08'
 '2015-07-09' '2015-07-10' '2015-07-11' '2015-07-12' '2015-07-13'
 '2015-07-14' '2015-07-15']
---------------------
['2015-07-04' '2015-07-03' '2015-07-02' '2015-07-01' '2015-06-30'
 '2015-06-29' '2015-06-28' '2015-06-27' '2015-06-26' '2015-06-25'
 '2015-06-24' '2015-06-23']
---------------------
227 days
```

1.7 fromfunction

- The fromfunction() function is used to construct an array by executing a function over each coordinate

[ ]:
```python
import numpy as np

x = np.fromfunction(lambda i:i**3, (10,))

print(x)
print('---------------------')
```

```
[  0.   1.   8.  27.  64. 125. 216. 343. 512. 729.]
---------------------
```

[ ]:
```python
import numpy as np

x = np.fromfunction(lambda i, j: i+j, (4, 5))

print(x)
print('---------------------')
```

```
[[0. 1. 2. 3. 4.]
 [1. 2. 3. 4. 5.]
 [2. 3. 4. 5. 6.]
 [3. 4. 5. 6. 7.]]
---------------------
```

[ ]:
```python
import numpy as np

x = np.fromfunction(lambda i, j, k: i+j+k, (2, 3, 4))

print(x)
print('---------------------')
```

```
[[[0. 1. 2. 3.]
  [1. 2. 3. 4.]
  [2. 3. 4. 5.]]

 [[1. 2. 3. 4.]
  [2. 3. 4. 5.]
  [3. 4. 5. 6.]]]
---------------------
```

[ ]:
```python
from numpy import *

def powers(i):
    i = i**2
    return i

x = fromfunction(powers, (9, ), dtype=int)

print(x)
```

```python
print('---------------------')
```

```
[ 0  1  4  9 16 25 36 49 64]
---------------------
```

[ ]:

```python
from numpy import *

m, n = 20, 5

def f(i):
    return (i % n == 0)

x = np.fromfunction(f, (m,), dtype=int)

print(x)
print('---------------------')
```

```
[ True False False False False  True False False False False  True False
 False False False  True False False False False]
---------------------
```

contact me Mahmoud Gadallah