```
# Come creare un numpy array: 1. Trasformando una lista di python base
2.Usare una funzione built-in       3. Generando numeri casuali


import numpy as np


# Creazione numpy array: conversione lista

# Lista unidimensionale

lista1 = [10,20,30,40]
lista1
```

```
[10, 20, 30, 40]
```

```
type(lista1)
```

```
list
```

```
lista1_array= np.array(lista1)
lista1_array
```

```
array([10, 20, 30, 40])
```

```
type(lista1_array)
```

```
numpy.ndarray
```

```
# Lista bidimensionale

lista2 = [[7,10,4],
          [40,50,60],
          [70,80,90],
          [32,11,32]]
lista2
```

```
[[7, 10, 4], [40, 50, 60], [70, 80, 90], [32, 11, 32]]
```

```
type(lista2)
```

```
list
```

```
lista2_array = np.array(lista2)
lista2_array
```

```
array([[ 7, 10,  4],
       [40, 50, 60],
       [70, 80, 90],
       [32, 11, 32]])
```

```
type(lista2_array)
```

```
numpy.ndarray
```

```
# Solitamente non si usano l liste per creare array numpy. Vediamo
quali built-in function sono più utilizzate

# Funzione arange: Va da n a n-1

array1 = np.arange(10)
array1
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
array2 = np.arange(4,10)
array2
```

```
array([4, 5, 6, 7, 8, 9])
```

```
array3 = np.arange(4,10,3)
array3
```

```
array([4, 7])
```

```
# 10 è escluso. Per includere il 10:

array3 = np.arange(4,11,3)
array3
```

```
array([ 4,  7, 10])
```

```
# Funzione zeros: Crea un array di zeri

zeri1 = np.zeros(10)
zeri1   # Crea un vettori che contiene 10 zeri
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
# Gli zeri creati finora sono di tipo float. Possiamo far si che siano
di tipo integer;


zeri1 = np.zeros(10,dtype=int)
zeri1
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
# Abbiamo finora creato un array di zeri unidimensionale. Creiamolo
ora bidimensionale (matrice)
```

```
zeri2_2D = np.zeros((10,4))
zeri2_2D
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
# Abbiamo così creato una matrice composta da 10 righe e 4 colonne.

# Come al solito gli zeri sono di tipo float. Modifichiamoli e
facciamoli diventare di tipo int

zeri2_2D_int = np.zeros((10,4),dtype=int)
zeri2_2D_int
```

```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

```
# np.ones. Funziona allo stesso modo di np.zeros, ma crea vettori o
matrici comoste da tutti 1 invece che 0


array_2D = np.ones((10,3),dtype = int)
array_2D
```

```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
```

```
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

```
# linspace. Si definisce un range di numeri, e poi il numero di numeri necessari per raggiungere l'estremo superiore (ricordiamo che è escluso)

array1 = np.linspace(2,23,10)
array1
```

```
array([ 2.        ,  4.33333333,  6.66666667,  9.        ,
11.33333333,
       13.66666667, 16.        , 18.33333333, 20.66666667,
23.        ])
```

```
# Creazione matrice diagonale tramite metodo eye. Ovviamente la matrice deve essere quadrata

matrix_1 = np.eye(5,5)
matrix_1
```

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

```
# Se la matrice NON è quadrata esce fuori una schifezza

matrix_1 = np.eye(5,2)
matrix_1
```

```
array([[1., 0.],
       [0., 1.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

```
# RANDOM
```

```
#random.rand. crea un vettore o una matrice avente numeri casuali tra
0 (incluso) e 1 (escluso) in base alla distribuzione uniforme



matrix_1 = np.random.rand(4,3)
matrix_1  #Matrice 4 righe e 3 colonne

array([[0.40645405, 0.0400123 , 0.40629821],
       [0.62661872, 0.87756145, 0.029362  ],
       [0.7429437 , 0.02106979, 0.92164692],
       [0.69579633, 0.61309203, 0.23258651]])



# La distribuzione uniforme è poco utilizzata. Solitamente si utilizza
la distribuzione normale standard, ossia la normale con media 0 e dev
standard 1.
# Solitamente i valori sono compresi tra -1 e 1, anche se possono
capitare valori fuori da questo range.
# Si utilizza quindi la funzione np.random.randn



matrix_1 = np.random.randn(3,4)
matrix_1

array([[-0.18958681, -1.84888103,  0.89045656,  0.64330036],
       [ 0.10296012, -0.68038152, -0.98981109,  0.11779583],
       [ 1.3208074 , -1.89767408, -0.85488977, -0.28068011]])



# RandInt

# Utilizzo di randint per creare un array unidimensionale composto da
16 valori. Ciascun valore sarà un numero intero compreso tra 2 e 10

matrix_1 = np.random.randint(2,10,16)
matrix_1

array([3, 9, 6, 5, 7, 7, 7, 6, 6, 9, 2, 9, 8, 2, 3, 4])

# Utilizzo di randint per creare una matrice avente 5 righe e 4
colonne. Ciascun valore sarà un numero intero compreso tra 2 e 10

matrix_2 = np.random.randint(2,10,(5,4))
matrix_2

array([[3, 4, 9, 5],
       [7, 4, 4, 3],
       [2, 9, 7, 6],
```

```
       [7, 6, 4, 8],
       [4, 8, 9, 2]])
```

```
# Ovviamente, ogni volta che eseguiamo riga 243 otterremo risulati
diversi. Per far si che ciò non accada, si può impostare un seed

np.random.seed(343)
np.random.randint(3,10,(4,4))
```

```
array([[4, 5, 7, 3],
       [9, 5, 9, 7],
       [7, 9, 4, 6],
       [6, 4, 5, 7]])
```

```
# Eseguendo più volte riga 291 otterremo sempre lo stesso risultato.
facciamo una prova senza seed

np.random.randint(3,10,(4,4))
```

```
array([[8, 6, 6, 5],
       [9, 7, 8, 7],
       [5, 6, 7, 5],
       [9, 5, 5, 7]])
```

```
np.random.randint(3,10,(4,4))
```

```
array([[7, 3, 9, 8],
       [8, 6, 6, 5],
       [6, 9, 8, 6],
       [3, 6, 5, 9]])
```

```
np.random.randint(3,10,(4,4))
```

```
array([[7, 9, 4, 9],
       [7, 8, 5, 7],
       [7, 5, 4, 5],
       [9, 4, 4, 9]])
```

```
# Facciamo ora una prova con seed

np.random.seed(343)
np.random.randint(3,10,(4,4))
```

```
array([[4, 5, 7, 3],
       [9, 5, 9, 7],
       [7, 9, 4, 6],
       [6, 4, 5, 7]])
```

```
np.random.seed(343)
np.random.randint(3,10,(4,4))
```

```
array([[4, 5, 7, 3],
       [9, 5, 9, 7],
       [7, 9, 4, 6],
       [6, 4, 5, 7]])
```

```
np.random.seed(343)
np.random.randint(3,10,(4,4))
```

```
array([[4, 5, 7, 3],
       [9, 5, 9, 7],
       [7, 9, 4, 6],
       [6, 4, 5, 7]])
```

```
# Senza seed i risultati cambiano, con seed non cambiano.
```

```
# reshape
```

```
arr= np.arange(4,100)
arr
```

```
array([ 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, 36,
37,
       38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
54,
       55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
71,
       72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88,
       89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
# Il numero di elementi è 100-4 = 96. Possiamo usare la funzione
reshape per passarer da un vettore ad una matrice
```

```
matrix = arr.reshape(24,4)
matrix
```

```
array([[ 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],
       [36, 37, 38, 39],
       [40, 41, 42, 43],
       [44, 45, 46, 47],
       [48, 49, 50, 51],
       [52, 53, 54, 55],
       [56, 57, 58, 59],
       [60, 61, 62, 63],
       [64, 65, 66, 67],
       [68, 69, 70, 71],
       [72, 73, 74, 75],
       [76, 77, 78, 79],
       [80, 81, 82, 83],
       [84, 85, 86, 87],
       [88, 89, 90, 91],
       [92, 93, 94, 95],
       [96, 97, 98, 99]])
```

```
# MAX
```

```
array_rand = np.random.randint(2,10,4)
array_rand
```

```
array([4, 5, 6, 4])
```

```
massimo = array_rand.max()
massimo
```

```
9
```

```
array_rand_matrix = array_rand = np.random.randint(2,15,(4,5))
array_rand_matrix
```

```
array([[10,  8,  7, 12, 11],
       [11,  8,  5, 12,  4],
       [ 5,  7, 12,  3, 14],
       [11,  6,  8, 10,  8]])
```

```
array_rand_matrix.max()
```

```
14
```

```
# MIN
```

```
array_rand = np.random.randint(2,10,4)
array_rand
```

```
array([8, 3, 9, 8])
```

```
minimo = array_rand.min()
minimo
```

```
3
```

```
array_rand_matrix = array_rand = np.random.randint(2,15,(4,5))
array_rand_matrix
```

```
array([[10,  8,  8,  6, 12],
       [ 3,  6, 12, 12,  4],
       [12,  5,  7, 14, 12],
       [ 2, 11,  2, 12, 11]])
```

```
array_rand_matrix.min()
```

```
2
```

```
# Per sapere la posizione nella quale si trova il numero massimo si
usa argmax, per sapere dove si trova il minimo si usa argmin
```

```
array_rand = np.random.randint(2,10,4)
array_rand
```

```
array([3, 4, 3, 9])
```

```
array_rand.argmax()
```

```
3
```

```
array_rand.argmin()
```

```
0
```

```
# dtype e shape
```

```
# dtype
```

```
array_rand = np.random.randint(2,10,4)
array_rand
```

```
array([8, 9, 6, 5])
```

```
array_rand.shape  # 1 riga, 4 colonne, ossia un vettore formato da 4 elementi
```

```
(4,)
```

```
array_rand.dtype # I valori sono tutti int32
```

```
dtype('int32')
```

```
# SELEZIONARE UN SINGOLO VALORE DA UN ARRAY UNIDIMENSIONALE
```

```
np.random.seed(29)
arr1 = np.random.randint(5,18,10)
arr1
```

```
array([10,  8, 17,  7, 13,  5, 14,  6, 13, 10])
```

```
arr1[2]
```

```
17
```

```
# SELEZIONARE UN SOTTOINSIEME DI ELEMENTI DA UN ARRAY UNIDIMENSIONALE

# Supponiamo di voler estrarre gli elementi 17,7,13,5,14. L'estremo inferiore è incluso, l'estremo superiore no
```

```
arr1[2:7]
```

```
array([17,  7, 13,  5, 14])
```

```
# ESTRAZIONE TUTTI GLI ELEMENTI FINO ALL'ELEMENTO 5
```

```
arr1[0:6]
```

```
array([10,  8, 17,  7, 13,  5])
```

```
# OPPURE
```

```
arr1[:6]
```

```
array([10,  8, 17,  7, 13,  5])
```

```
# ESTRAZIONE ELEMENTI DALL'ELEMENTO 7 FINO ALLA FINE DEL VETTORE:
```

```
arr1[3:]
```

```
array([ 7, 13,  5, 14,  6, 13, 10])
```

```
# DIFFERENZA TRA ARRAY PYTHON BASE E NUMPY ARRAY: I numpy array possono attuare il broadcasting, ossia è possibile manipolare l'array di partenza. Facciamo un esempio.
```

```
arr1
```

```
array([10,  8, 17,  7, 13,  5, 14,  6, 13, 10])
```

```
arr1[4]=100
```

```
arr1
```

```
array([ 10,   8,  17,   7, 100,   5,  14,   6,  13,  10])
```

```
# POSSIAMO ANCHE MODIFICARE PIU DI UN ELEMENTO. Ad esempio, supponiamo di voler modificare gli elementi dalla posizione 4 alla posizione 7:
```

```
arr1
```

```
array([ 10,   8,  17,   7, 100,   5,  14,   6,  13,  10])
```

```
arr1[4:8] = 2340
```

```
arr1
```

```
array([  10,    8,   17,    7, 2340, 2340, 2340, 2340,   13,   10])
```

```
# OPPURE MODIFICHIAMO GLI ULTIMI 3 ELEMENTI:
```

```
arr1[-3:] = 0
```

```
arr1
```

```
array([  10,    8,   17,    7, 2340, 2340, 2340,    0,    0,    0])
```

```
# NOTIAMO ORA UN FENOMENO MOLTO IMPORTANTE. RIPRENDIAMO IL NOSTRO ARRAY DI PARTENZA
```

```
np.random.seed(29)
arr1 = np.random.randint(5,18,10)
arr1
```

```
array([10,  8, 17,  7, 13,  5, 14,  6, 13, 10])
```

```
# ESTRAIAMO UN SOTTOINSIEME DI QUESTO ARRAY ED INSERIAMOLO IN UNA VARIABILE

arr2 = arr1[3:8]
arr2
```

```
array([ 7, 13,  5, 14,  6])
```

```
# MODIFICHIAMO arr2


arr2[:] = 18
arr2
```

```
array([18, 18, 18, 18, 18])
```

```
# COSA SUCCEDE AD arr1?

arr1
```

```
array([10,  8, 17, 18, 18, 18, 18, 18, 13, 10])
```

```
# SI E' MODIFICATO ANCH'ESSO. COME POSSIAMO RISOLVERE QUESTO PROBLEMA? CREANDO UNA COPIA DI arr1

np.random.seed(29)
arr1 = np.random.randint(5,18,10)
arr1
```

```
array([10,  8, 17,  7, 13,  5, 14,  6, 13, 10])
```

```
arr11 = arr1.copy()
arr11
```

```
array([10,  8, 17,  7, 13,  5, 14,  6, 13, 10])
```

```
arr2 = arr11[3:8]
arr2
```

```
array([ 7, 13,  5, 14,  6])
```

```
arr2[:] = 18
arr2
```

```
array([18, 18, 18, 18, 18])
```

```
# arr1 E' STATO MODIFICATO?
```

```
arr1
```

```
array([10,  8, 17,  7, 13,  5, 14,  6, 13, 10])
```

```
# NO! PROBLEMA RISOLTO.
```

```
# ARRAY BIDIMENSIONALI (MATRICI)
```

```
np.random.seed(29)
M = np.random.randint(5,18,(6,4))
M
```

```
array([[10,  8, 17,  7],
       [13,  5, 14,  6],
       [13, 10,  8,  6],
       [13,  6, 16, 16],
       [10,  9, 12,  5],
       [ 9,  7, 11, 12]])
```

```
# PER CONTARE QUANTE RIGHE E QUANTE COLONNE HA UNA MATRICE SI USA LA FUNZIONE SHAPE
```

```
M.shape
```

```
(6, 4)
```

```
# ESTRAZIONE PRIMA RIGA
```

```
M[0]
```

```
array([10,  8, 17,  7])
```

```
# ESTRAZIONE DA RIGA 2 A RIGA 5
```

```
M[1:5]
```

```
array([[13,  5, 14,  6],
       [13, 10,  8,  6],
```

```
       [13,  6, 16, 16],
       [10,  9, 12,  5]])
```

```
# ESTRAZIONE COLONNA 2
```

```
M[:,1]
```

```
array([ 8,  5, 10,  6,  9,  7])
```

```
# ESTRAZIONE COLONNE 2 E 3
```

```
M[:,1:3]
```

```
array([[ 8, 17],
       [ 5, 14],
       [10,  8],
       [ 6, 16],
       [ 9, 12],
       [ 7, 11]])
```

```
# ESTRAZIONE RIGHE DA 2 A 4 E COLONNA 1 E 2
```

```
M[1:4,1:3]
```

```
array([[ 5, 14],
       [10,  8],
       [ 6, 16]])
```

```
# SUBSET TRAMITE MASCHERA BOOLEANA

# SUPPONIAMO DI VOLER CAPIRE QUALI VALORI DI M SONO < 10.
```

```
M < 10
```

```
array([[False,  True, False,  True],
       [False,  True, False,  True],
       [False, False,  True,  True],
       [False,  True, False, False],
       [False,  True, False,  True],
       [ True,  True, False, False]])
```

```
# ABBIAMO TRUE QUANDO IL VALORE E' <10, FALSE IN CASO CONTRARIO.
SOLITAMENTE QUESTO RISULTATO SI INSERISCE IN UNA VARIABILE CHIAMATA
mask:
```

```
mask = M < 10

# INFINE POSSIAMO FILTRARE:

M[mask]

array([8, 7, 5, 6, 8, 6, 6, 9, 5, 9, 7])

# ECCO TROVATI TUTTI I NUMERI < 10. ESSI FORMERANNO UN ARRAY
UNIDIMENSIONALE



# OPERAZIONI TRA ARRAY

# Mentre le operazioni tra scalari sono sempre possibili, le
operazioni tra vettori e/o matrici sono possibili solo se entrambi i
vellori o le matrici hanno le stesse dimensioni.
# Consideriamo due matrici

np.random.seed(10)
M1 = np.random.randint(0,3,(2,3))

np.random.seed(20)
M2 = np.random.randint(15,66,(2,3))

M1

array([[1, 1, 0],
       [0, 1, 0]])

M2

array([[50, 41, 30],
       [46, 43, 41]])



# Somma matrice per scalare

M1 + 4

array([[5, 5, 4],
       [4, 5, 4]])

# Differenza matrice per scalare

M1 - 4

array([[-3, -3, -4],
       [-4, -3, -4]])

# Moltiplicazione matrice per scalare
```

```
M1 * 4
```

```
array([[4, 4, 0],
       [0, 4, 0]])
```

```
# Divisione matrice per scalare
```

```
M1 / 4
```

```
array([[0.25, 0.25, 0.  ],
       [0.  , 0.25, 0.  ]])
```

```
# DIVISIONE PER 0
```

```
4/ M1
```

```
C:\Users\ACER\AppData\Local\Temp\ipykernel_33984\2221949893.py:1:
RuntimeWarning: divide by zero encountered in divide
  4/ M1
```

```
array([[ 4.,  4., inf],
       [inf,  4., inf]])
```

```
# Mentre con python base avremo un errore, con numpy avremo solo un
warning e il risultato è infinito, così come insegna l'analisi
matematica.
```

```
# SOMMA TRA MATRICI
```

```
M1+M2
```

```
array([[51, 42, 30],
       [46, 44, 41]])
```

```
# DIFFERENZA TRA MATRICI
```

```
M1-M2
```

```
array([[-49, -40, -30],
       [-46, -42, -41]])
```

```
# MOLTIPLICAZIONE TRA MATRICI
```

```
M1*M2
```

```
array([[50, 41,  0],
       [ 0, 43,  0]])
```

```
# DIVISIONE TRA MATRICI
```

```
M1/M2
```

```
array([[0.02      , 0.02439024, 0.        ],
       [0.        , 0.02325581, 0.        ]])
```

```
# FUNZIONI IN-BUILT TRA VETTORI E MATRICI. Vedremo queste funzioni
applicate ad una matrice, ma funzionano anche sui vettori
```

```
np.random.seed(10)
M1 = np.random.randint(3,16,(2,3))
M1
```

```
array([[12,  7,  3],
       [ 4, 14, 15]])
```

```
M1.sum()     # Somma generale
```

```
55
```

```
M1.mean()    # Media generale
```

```
9.166666666666666
```

```
M1.max()    # Massimo
```

```
15
```

```
M1.min()    # Minimo
```

```
3
```

```
M1.var()    # Varianza generale
```

```
22.472222222222218
```

```
M1.std()   # Deviazione standard generale
```

```
4.740487551109297
```

```
M1.sum(axis=0)    # Somma lungo le righe
```

```
array([16, 21, 18])
M1.sum(axis=1)    # Somma lungo le colonne
array([22, 33])

M1.mean(axis=0)  # Media scorrendo le righe
array([ 8. , 10.5,  9. ])
M1.mean(axis=1)  # Media scorrendo le colonne
array([ 7.33333333, 11.        ])
```