Gli array numpy sono dei container con un'ampiezza fissa contenenti elementi. Rispetto alle liste di python base o alle tuple sono più efficienti nel processare dati. In particolare:

- Possono immagazzinare un solo tipo di dato (Tipi di dati di vario tipo possono essere immagazzinati come stringhe)
- Possono essere unidimensionali o multidimensionali
- Gli elemeenti presenti nell'array possono essere modificati, ma il numero di elementi è fisso

```

```

Partiamo da una lista in python base e trasformiamola in un array pandas:

```
import numpy as np

animali = ["canarino","culbianco","capibara","cinciallegra"]

animali_array = np.array(animali)

animali_array
```

```
array(['canarino', 'culbianco', 'capibara', 'cinciallegra'],
dtype='<U12')
```

Le proprietà chiave degli array pandas sono:

- ndim ==> Il numero delle dimensioni degli array
- shape ==> La lunghezza di ciascuna dimensione dell'array
- size ==> il numero totale di elementi presenti in un array
- dtype ==> Il tipo di dato degli elementi presenti nell'array

```

```

Ripartiamo dal nostro array animali_array

```
animali_array
```

```
array(['canarino', 'culbianco', 'capibara', 'cinciallegra'],
dtype='<U12')
```

```
type(animali_array)
```

```
numpy.ndarray
```

ndarray è un tipo di dato che significa n-dimensional array

```
animali_array.ndim
```

```
1
```

```
animali_array.shape
```

```
(4,)
```

```
animali_array.size
```

```
4
```

```
animali_array.dtype
```

```
dtype('<U12')
```

Spieghiamo le funzioni viste:

- La proprietà ndim ci dice che animali_array ha una sola dimensione
- La proprietù shape ci dice che la dimensione del nostro array è composta da quattro elementi
- La proprietà size ci dice che, in totale, il nostro array ha 4 elementi
- La proprietà dtype ci dice che il tipo di tutti i dati presenti nel nostro array è <U12'. Questo tipo di dato è presente solo in numpy

Passiamo ora ad un array bidimensionale. Supponiamo di avere una lista di animali posseduti da Gino, una lista di animali posseduti da Fabrizio e una lista per Ascanio. Si ricordi che, affinche sia possibile creare un array, gli elementi presenti in ciascua lista devono essere di pari numero

```
animali_gino = ["cane","gatto"]
```

```
animali_fabrizio = ["capibara","casatoro"]
```

```
animali_ascanio = ["culbianco","kiwi"]
```

```
animali_studenti = [animali_gino,animali_fabrizio,animali_ascanio]
```

```
animali_studenti
```

```
[['cane', 'gatto'], ['capibara', 'casatoro'], ['culbianco', 'kiwi']]
```

```
animali_studenti_array = np.array(animali_studenti)

animali_studenti_array

array([['cane', 'gatto'],
       ['capibara', 'casatoro'],
       ['culbianco', 'kiwi']], dtype='<U9')
```

Vediamone le proprietà

```
animali_studenti_array.ndim

2


animali_studenti_array.shape

(3, 2)


animali_studenti_array.size

6


animali_studenti_array.dtype
```

Quindi, un array di tipo numpy è una lista base di n liste base di python

```

```

Riprendiamo in considerazione i nostri animali_gino, animali_fabrizio, animali_ascanio

```
# Creazione array unidimensionale con animali_gino:

animali_gino_array = np.array(animali_gino)

animali_gino_array

array(['cane', 'gatto'], dtype='<U5')
```

```
# A partire da un array bidimensionale, la sintassi cambia leggermente

animali_gino_fabrizio_array =
np.array([animali_gino,animali_fabrizio])

animali_gino_fabrizio_array
```

```
array([['cane', 'gatto'],
       ['capibara', 'casatoro']], dtype='<U8')
```

In sintesi, se vogliamo creare un array di tipo numpy a partire da due o più liste, siamo costretti ad inserire tali liste tra parentesi quadre

FUNZIONE RANGE

Quando si lavora, può convenire fare delle prove creando una sequenza di numeri casuali, invece di creare gli array ogni volta. Uno di questi modi è l'utilizzo della funzione range

```
range(6)
```

```
range(0, 6)
```

```
array_uni= np.array(range(4))

array_uni
```

```
array([0, 1, 2, 3])
```

```
array_bi = np.array([range(4),range(4)])

array_bi
```

```
array([[0, 1, 2, 3],
       [0, 1, 2, 3]])
```

```
array_bi + 5
```

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

```
0.5*array_bi
```

```
array([[0. , 0.5, 1. , 1.5],
       [0. , 0.5, 1. , 1.5]])
```

```
array_bi + 5 + array_bi
```

```
array([[ 5,  7,  9, 11],
       [ 5,  7,  9, 11]])
```

E' quindi possibile effettuare calcoli sugli array numpy

PROPRIETA FONDAMENTALI DEGLI ARRAY array_b: ndim,shape,size,dtype

```
array_bi.ndim
```

```
2
```

```
array_bi.shape
```

```
(2, 4)
```

```
array_bi.size
```

```
8
```

```
array_bi.dtype
```

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

TRASPOSIZIONE

```
# Riprendiamo il nostro array:
array_bi
```

```
array([[0, 1, 2, 3],
       [0, 1, 2, 3]])
```

```
# con la seguente funzione possiamo trasporlo:
array_bi_trasposto = array_bi.T

array_bi_trasposto
```

```
array([[0, 0],
       [1, 1],
       [2, 2],
       [3, 3]])
```

```
array_bi_trasposto.shape
```

```
(4, 2)
```

Il nuovo array ha infatti 4 righe e 2 colonne.

Finora abbiamo sempre utilizzato il termine generico array. Entriamo più nel dettaglio

- UN ARRAY UNIDIMENSIONALE NUMPY è CHIAMATO ARRAY
- UN ARRAY n-dimensionale numpy, con n > 1, è chiamato MATRICE

Continuiamo a vedere le nostre proprietà fondamentali della nostra matrice array_bi_trasposto

```
array_bi_trasposto.ndim
```

```
2
```

```
array_bi_trasposto.size
```

```
8
```

```
array_bi_trasposto.dtype
```

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

ESERCIZIO 1

NumPy Arrays A NumPy array, 'age_array' has been created from a list of ages.

Store the size of the array in the variable 'size', and the shape of the array in the variable 'shape', then print both variables.

SOLUZIONE ESERCIZIO 1

```
import numpy as np

age_list = [22, 34, 57, 65, 87, 19, 44]

age_array = np.array(age_list)

size = age_array.size
shape = age_array.shape

print(size)
print(shape)
```

```
7
(7,)
```