# 05. Introduzione a NumPy

Corso di Python per il Calcolo Scientifico

Angelo Cardellicchio
angelo.cardellicchio@stiima.cnr.it

# Outline

- Array e NumPy
- Operazioni fondamentali sugli array
- Algebra
- Polinomi
- Statistica

# Array (1)

- Gli **array** sono la struttura dati principale di NumPy
- Rappresentano un generico **tensore**
- Sono composti da **elementi omogenei** (ovvero dello stesso tipo)
- Possono essere sia monodimensionali, sia n-dimensionali

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

- Possiamo valutarne le dimensioni mediante la proprietà **`shape`**

```
a.shape
```

# Array (2)

- Esistono diversi metodi per la creazione rapida di un array

```
u = np.ones(shape=(3,3))              # creo un array di tutti 1
z = np.zeros(shape=(3,3))   # creo un array di tutti 0
e = np.empty(shape=(3,3))   # creo un array non inizializzato
i = np.eye(3)                         # creo la matrice identità
```

- Il metodo **`diag(x)`** permette di:
  - estrarre la diagonale se **x** è una matrice;
  - ottenere una matrice diagonale con **x** diagonale se **x** è un vettore.
- Possiamo anche ottenere matrici triangolari inferiori e superiori usando i metodi **`tril`** e **`triu`**.

# Array (3)

- Esistono diversi metodi per la creazione rapida di un array

```
u = np.ones(shape=(3,3))          # creo un array di tutti 1
z = np.zeros(shape=(3,3))         # creo un array di tutti 0
e = np.empty(shape=(3,3))         # creo un array non inizializzato
i = np.eye(3)                     # creo la matrice identità
```

- Il metodo **`diag(x)`** permette di:
  - estrarre la diagonale se **x** è una matrice;
  - ottenere una matrice diagonale con **x** diagonale se **x** è un vettore.

# Array (4)

- L'accesso agli elementi di un array avviene mediante indicizzazione (proprio come per le liste)

```
a = np.array([1, 2, 3, 4])    # a[0] è 1
```

- È anche possibile usare delle maschere booleane:

```
b = np.array([[1,2], [3,4]])
mask = ([True, False], [True, False])
b[mask]                       # array([1, 3])
```

- Anche gli array prevedono le operazioni di slicing

# Operazioni fondamentali sugli array (1)

- È possibile effettuare tutte le operazioni aritmetiche fondamentali **elemento per elemento**
- È importante che le dimensioni degli array siano coerenti

```
a = np.array([1, 2])
b = np.array([3, 4])
a + b                                   # array([4, 6])
```

- La funzione sum permette di sommare tutti gli elementi di un array lungo un certo asse (di default, le righe)

```
b = np.array([[1, 2], [3, 4]])
b.sum(axis=0)                           # array([4, 6])
```

# Operazioni fondamentali sugli array (2)

- La funzione **`dot()`** permette di effettuare una moltiplicazione matriciale.

```
a = np.array([[1, 2]])
b = np.array([[3], [4]])
a.dot(b)                                    # array([11])
```

- La funzione **`sort()`** ordina gli elementi di un array. È possibile specificare l'asse di ordinamento.

```
mat = np.array([[2,3,1], [4,2,6], [7,5,1]])
np.sort(mat, axis=0)
# array([[2, 2, 1], [4, 3, 1], [7, 5, 6]])
```

# Operazioni fondamentali sugli array (3)

- La funzione **`concatenate()`** ci permette di concatenare due array.

```
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
np.concatenate((a, b))                  # array([1, 2, 3, 4, 5, 6, 7, 8])
```

- Le funzioni **`delete()`** ed **`insert()`** ci permettono di rimuovere ed inserire un elemento in un array, mentre la funzione **`append()`** ci permette di inserire i valori specificati in coda all'array.

```
arr = np.array([1, 2, 3, 4])
np.delete(arr, 0)           # array([2, 3, 4])

np.insert(mat, 3, [10, 11, 12], 0)
# array([[ 1, 2, 3], [ 4, 5, 6], [ 7, 8, 9], [10, 11, 12]])
```

# Operazioni fondamentali sugli array (4)

- Le proprietà **`ndim`**, **`size`** e **`shape`** ci permettono di descrivere, rispettivamente, il numero di dimensioni di un array, il numero di elementi e la dimensionalità di un array.
- Per modificare le dimensioni di un array usiamo la funzione **`reshape()`**.

```
mat = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,16]])
mat.reshape((2, 8))
# array([[ 1, 2, 3, 4, 5, 6, 7, 8], [ 9, 10, 11, 12, 13, 14, 15, 16]])
```

- È possibile vettorizzare un array mediante la funzione **`flatten()`**.

```
mat.flatten()                    # array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- Si può anche usare la funzione **`ravel()`**, che non restituisce una copia come la **`flatten()`**, ma agisce sull'array iniziale.

# Domande?

42