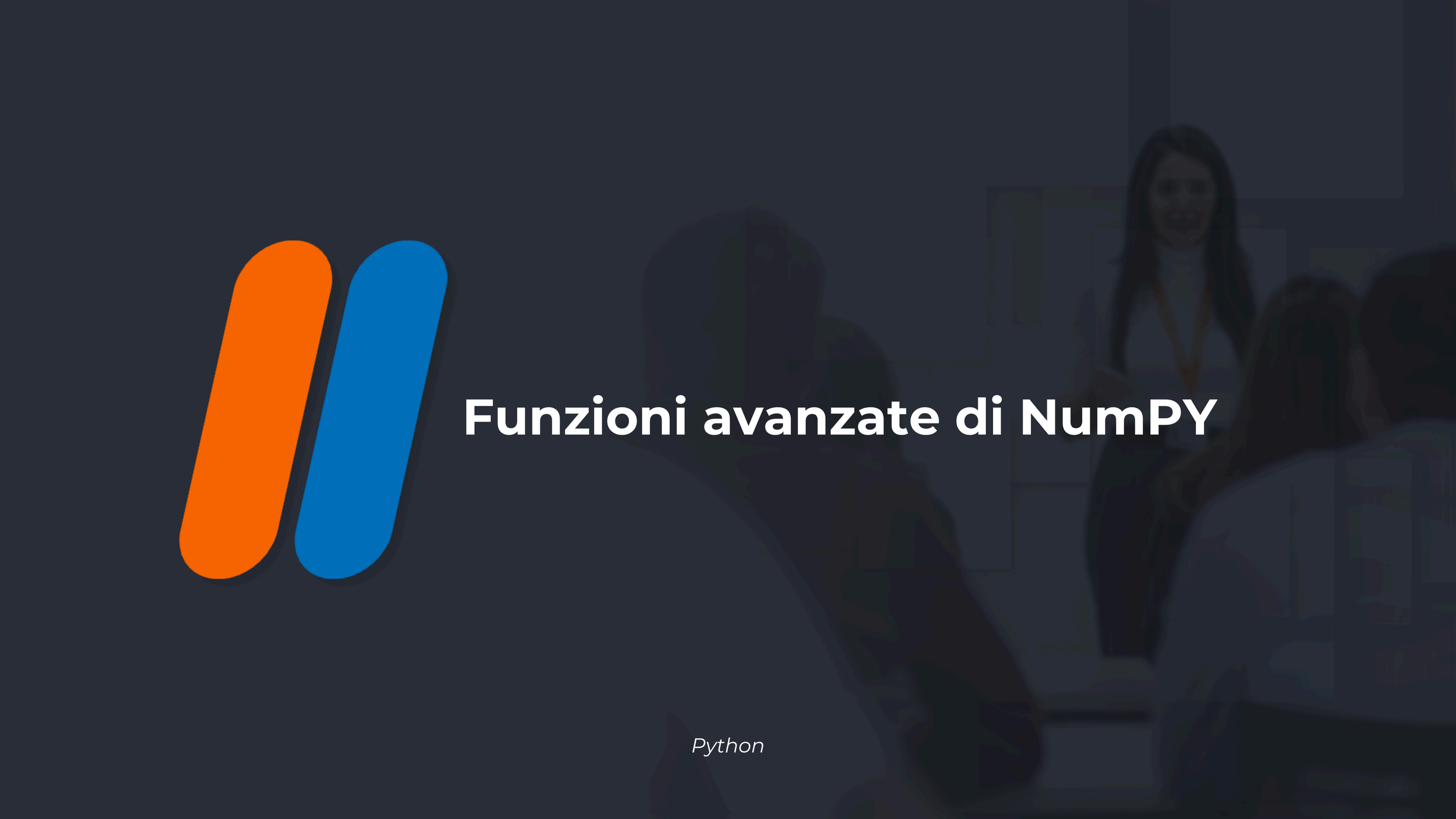


Funzioni avanzate di NumPY

Python

Broadcasting

Il broadcasting è una potente funzionalità di NumPy che permette di eseguire operazioni aritmetiche tra array di dimensioni diverse senza dover fare esplicito "broadcasting" dei dati.

Quando due array hanno forme diverse, NumPy tenta di allinearli secondo regole specifiche in modo da rendere le operazioni elementari (somma, sottrazione, ecc.) possibili senza dover creare copie ridondanti dei dati.

Questo aumenta significativamente l'efficienza e riduce l'uso della memoria.

Broadcasting Scopo

Il broadcasting consente di eseguire operazioni tra array di dimensioni diverse in modo efficiente, evitando la necessità di espandere manualmente gli array per farli combaciare.

Questa funzionalità è essenziale per lavorare con dati multidimensionali in modo conciso e performante.

Broadcasting Errori Comuni:

Un errore comune è supporre che il broadcasting avvenga in tutte le situazioni.

Se le dimensioni degli array non sono compatibili secondo le regole del broadcasting, si ottiene un errore ValueError.

Inoltre, può essere difficile per i principianti comprendere come funziona il broadcasting in casi complessi, portando a risultati inattesi o difficili da interpretare.

Broadcasting Es:

```
import numpy as np

# Creazione di un array 2x3
A = np.array([[1, 2, 3], [4, 5, 6]])

# Creazione di un array 1x3
B = np.array([10, 20, 30])

# Operazione di somma utilizzando il broadcasting
# B viene "broadcasted" su ciascuna riga di A
C = A + B

print(C)
# Output:
# [[11 22 33]
#  [14 25 36]]
```

In questo esempio, l'array B (1x3) viene automaticamente esteso su ogni riga di A (2x3) grazie al broadcasting, consentendo una somma elemento per elemento senza dover ridimensionare manualmente B.

Vettorializzazione

La vettorializzazione è il processo di conversione delle operazioni scalari (che agiscono su singoli valori) in operazioni vettoriali (che agiscono su interi array).

NumPy supporta nativamente operazioni vettorializzate, permettendo di eseguire calcoli complessi su grandi quantità di dati senza dover utilizzare cicli espliciti in Python.

Questo approccio non solo rende il codice più conciso, ma è anche molto più veloce grazie all'ottimizzazione interna di NumPy che sfrutta librerie come BLAS e LAPACK.

Vettorializzazione Scopo

La vettorializzazione mira a ottimizzare le operazioni su array utilizzando operazioni nativamente supportate da NumPy, senza ricorrere a cicli espliciti.

Questo approccio riduce significativamente il tempo di esecuzione del codice e semplifica la scrittura di algoritmi complessi.

Vettorializzazione Errori Comuni:

Gli errori comuni includono il tentativo di applicare funzioni non vettorializzate a array NumPy, il che può portare a errori di tipo o a prestazioni inferiori.

Un altro errore frequente è la mancanza di consapevolezza su come le operazioni vettorializzate gestiscono i dati, portando a risultati inattesi se i dati di input non sono correttamente preparati.

Vettorializzazione Es:

```
1. import numpy as np
2.
3. # Creazione di un array di numeri da 0 a 999
4. x = np.arange(1000)
5.
6. # Calcolo del quadrato di ciascun elemento con vettorializzazione
7. y = x ** 2
8.
9. print(y[:10])
10. # Output:
11. # [ 0  1  4  9 16 25 36 49 64 81]
```

Memmap

Il memmap è una funzionalità di NumPy che consente di mappare grandi file binari su disco in array NumPy senza caricare interamente il file in memoria.

Questo è particolarmente utile quando si lavora con dataset molto grandi che non possono essere caricati completamente in RAM.

Il memmap permette di accedere ai dati come se fossero un normale array NumPy, ma i dati vengono caricati in memoria solo quando necessario, riducendo il consumo di memoria e migliorando l'efficienza.

Memmap Scopo

Il memmap consente di lavorare con file di grandi dimensioni senza caricarli completamente in memoria, mappandoli direttamente su un array NumPy.

Questo è particolarmente utile per analizzare grandi dataset che non possono essere gestiti interamente in RAM, migliorando l'efficienza dell'uso della memoria

Memmap Errori Comuni

Gli errori comuni con memmap includono la perdita di dati dovuta a operazioni di scrittura non salvate correttamente su disco, e il mancato rilascio di file memorizzati (chiusura del file) che può portare a problemi di blocco del file o esaurimento delle risorse.

Inoltre, poiché il memmap non è adatto per accessi casuali molto frequenti, un uso improprio può portare a rallentamenti significativi nelle prestazioni.

Memmap (Memory-mapped files) Es:

```
1. import numpy as np
2.
3. # Creazione di un file memorizzato in memoria per scrittura
4. fp = np.memmap('large_file.dat', dtype='float32', mode='w+', shape=(10000, 10000))
5.
6. # Scrittura di dati nel file memorizzato in memoria
7. fp[:,:] = np.random.rand(10000, 10000)
8.
9. # Accesso ai dati senza caricare tutto in memoria
10. subset = fp[5000:5010, 5000:5010]
11.
12. print(subset)
13.
14. # Chiusura del file
15. del fp  # Importante per assicurarsi che i dati siano scritti su disco
```

Questo esempio mostra come utilizzare memmap per gestire un array di grandi dimensioni senza caricarlo completamente in memoria.

Il file large_file.dat viene creato e popolato con numeri casuali, e un piccolo sottoinsieme viene letto successivamente. Il comando del fp è importante per garantire che tutte le scritture vengano effettivamente salvate su disco.

Strutture Dati Multidimensionali Avanzate

Le Strutture Dati Multidimensionali Avanzate o meglio gli structured arrays e i record arrays in NumPy permettono di gestire dati eterogenei, dove ogni elemento dell'array può essere una tupla di variabili con tipi diversi.

Questa funzionalità è utile per lavorare con dataset complessi che contengono diversi tipi di dati (ad esempio, numeri, stringhe, ecc.).

I record arrays, una variante degli structured arrays, offrono un accesso facilitato ai campi tramite attributi, rendendo la manipolazione dei dati più intuitiva e simile all'accesso agli attributi di un oggetto.

Strutture Dati Multidimensionali Avanzate Scopo

Le strutture dati avanzate come i structured arrays e i record arrays permettono di gestire dataset complessi e eterogenei all'interno di NumPy, dove ogni elemento può avere tipi di dati diversi.

Questo è utile in situazioni in cui si devono manipolare dati strutturati simili a quelli che si trovano comunemente nei database o nei file CSV.

Strutture Dati Multidimensionali Avanzate Errori Comuni:

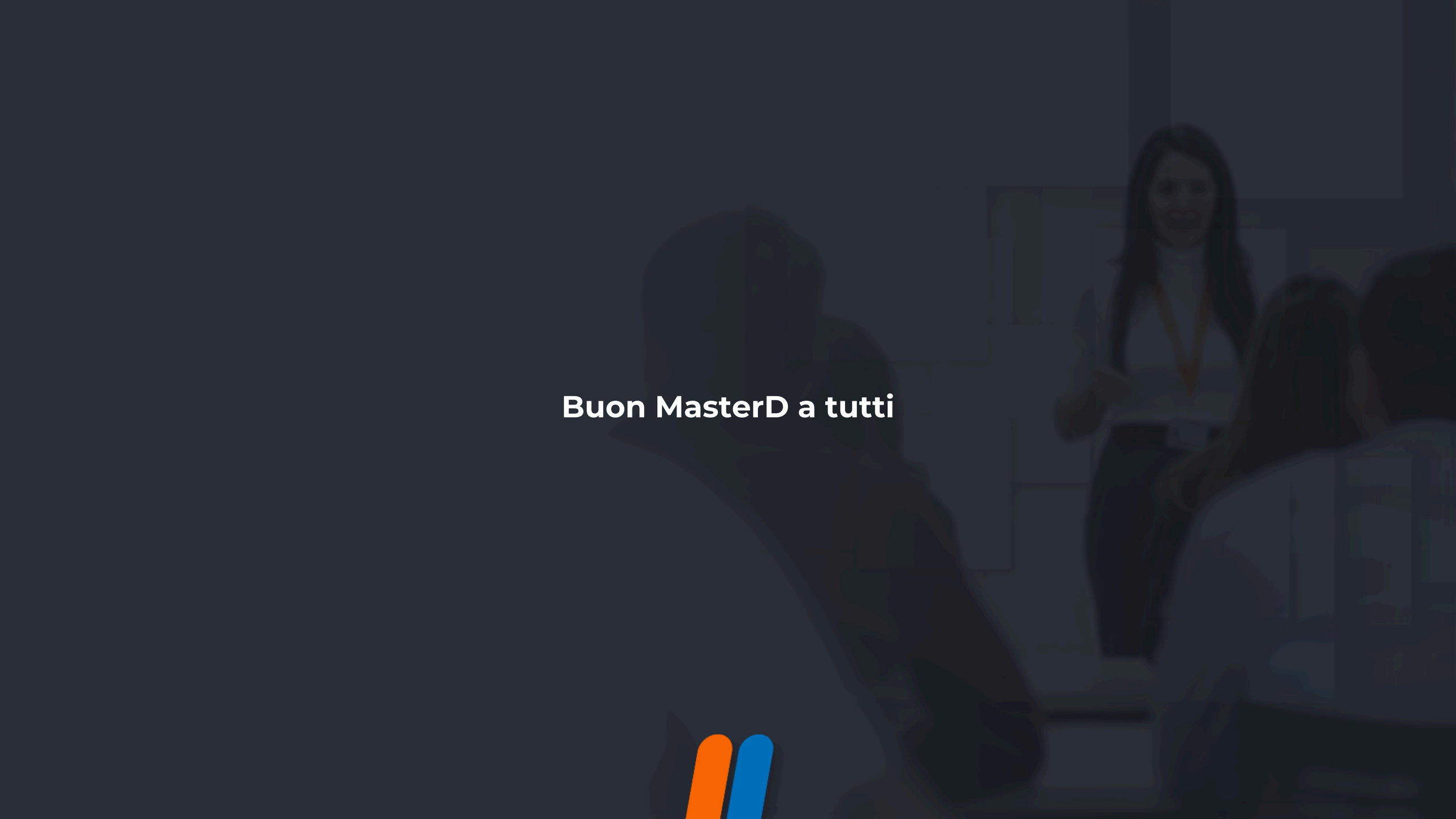
Gli errori comuni includono la confusione tra gli array strutturati e i semplici array multidimensionali, il che può portare a operazioni errate sui dati.

Un altro errore frequente è dimenticare di definire correttamente i nomi e i tipi di campo, il che può portare a difficoltà nell'accesso ai dati o addirittura a errori di tipo durante l'esecuzione delle operazioni.

Strutture Dati Multidimensionali Avanzate Es:

```
1. import numpy as np
2.
3. # Definizione di un array strutturato con campi di tipo diverso
4. person_dtype = np.dtype([('name', 'S10'), ('age', 'i4'), ('weight', 'f4')])
5.
6. # Creazione di un array strutturato
7. people = np.array([('Alice', 25, 55.0), ('Bob', 30, 85.5)], dtype=person_dtype)
8.
9. # Accesso ai dati tramite i nomi dei campi
10. print(people['name'])
11. # Output: [b'Alice' b'Bob']
12.
13. # Accesso ai singoli record
14. print(people[1])
15. # Output: (b'Bob', 30, 85.5)
```

Qui viene creato un array strutturato che contiene informazioni su persone, inclusi nome, età e peso.
I dati possono essere facilmente accessibili tramite i nomi dei campi.
Questo approccio è molto utile per lavorare con dataset eterogenei in cui ogni elemento ha più attributi di diverso tipo.


Buon MasterD a tutti