



# Introduzione a Python e OpenCV

Sistemi Digitali

A.A. 2024/2025

Fabio Tosi

Università di Bologna



**NOTA IMPORTANTE**



“Questo materiale **NON** fa parte del programma del corso e **NON** sarà richiesto all'esame. È fornito come supporto per studenti interessati a sviluppare progetti in Python per l'esame, per chi vuole approfondire CUDA in Python utilizzando librerie come cuPy, PyCUDA o framework di alto livello come PyTorch, o per coloro che desiderano muovere i primi passi nella computer vision con OpenCV in Python.”

# **Python: Una Guida Rapida**

# Introduzione a Python

## Cosa è Python?

- Python è un **linguaggio di programmazione di alto livello**, ideato da Guido van Rossum nel 1991, e oggi mantenuto da una vasta comunità di sviluppatori e organizzazioni.
- È un **linguaggio orientato agli oggetti**, ma supporta anche altri paradigmi come la programmazione **procedurale, funzionale e scripting**.
- Ha una **tipizzazione dinamica**, il che significa che i tipi di dati sono determinati a runtime, rendendolo flessibile ma richiedendo attenzione nella gestione dei dati.
- Python è spesso definito un **linguaggio interpretato**: il codice sorgente viene tradotto in **bytecode**, un formato intermedio, che viene poi eseguito dall'interprete Python.

## Aziende che utilizzano Python



Uber

Google



Instagram



Quora

amazon

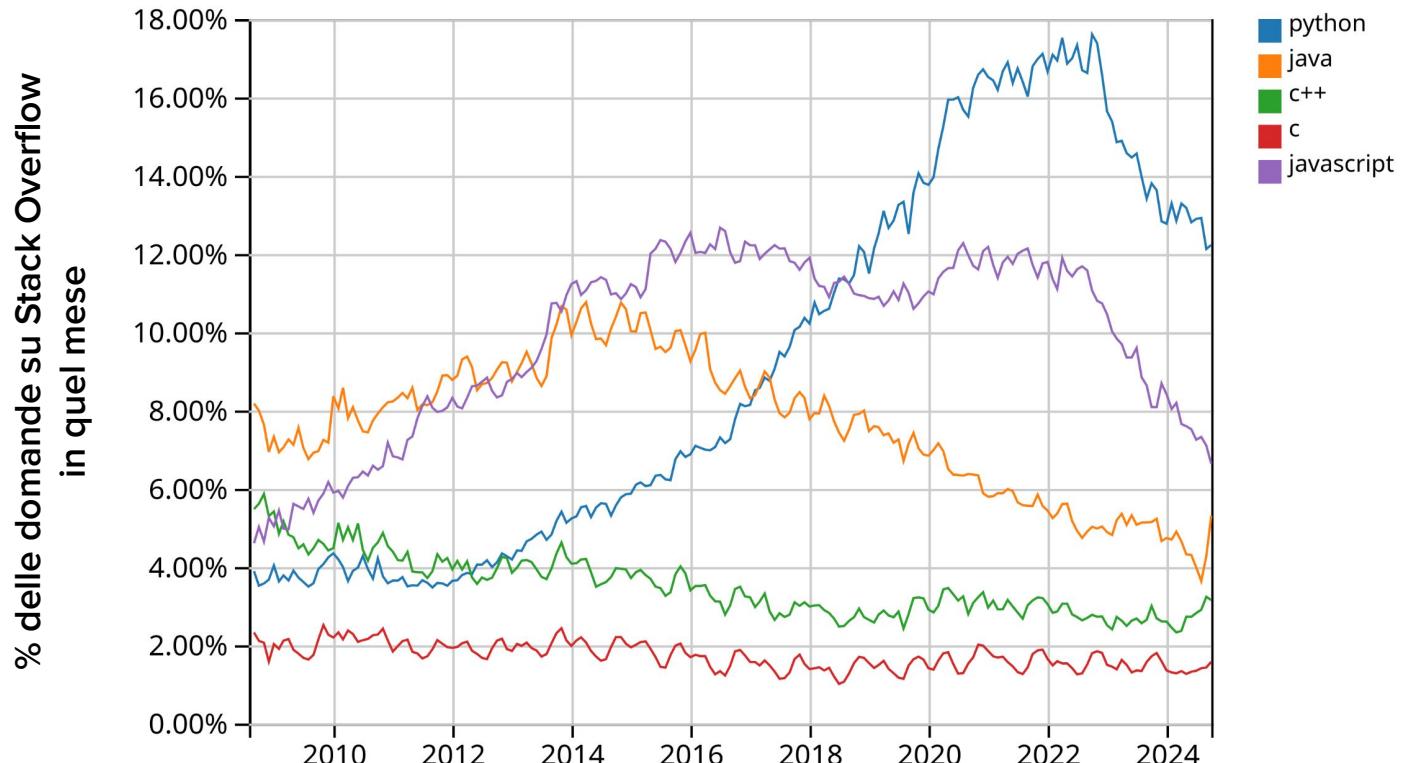


reddit



# Introduzione a Python

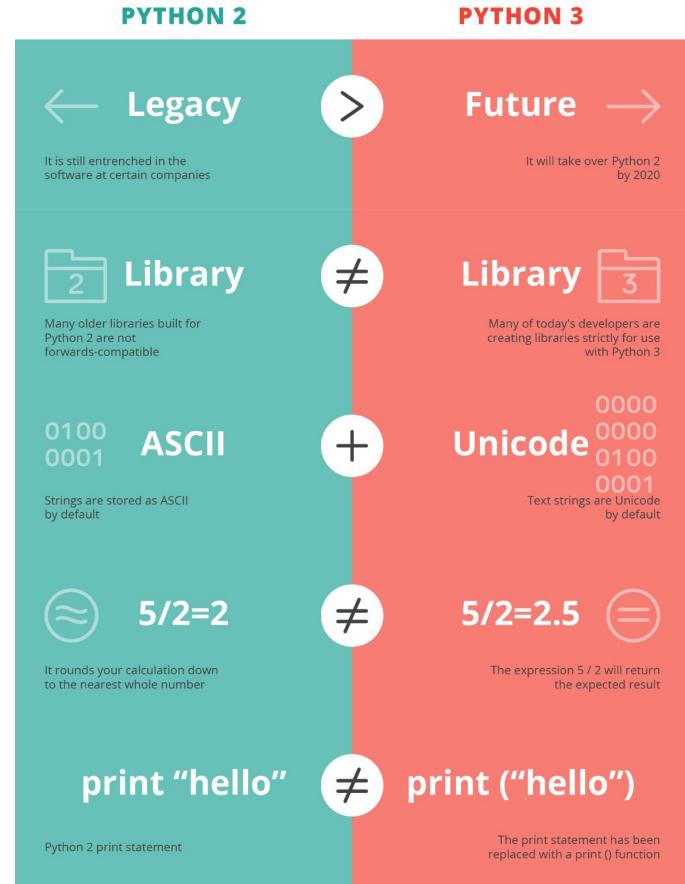
- Tendenza dei linguaggi di programmazione (ultimo aggiornamento 18/08/2024) su [StackOverflow](#)



# Introduzione a Python

## Versioni

- Esistono due versioni principali di Python:
  - Python 2.x
  - Python 3.x
- Sebbene molto simili, il codice Python 2.x potrebbe non essere compatibile con 3.x e viceversa.
- La maggior parte delle librerie e dei framework (ad esempio, NumPy) sono disponibili per entrambe le versioni.
- Il supporto per Python 2.x è **cessato** a gennaio 2020.



# Setup Python: Virtualenv

## Cos'è virtualenv?

- Strumento per creare **ambienti Python isolati**.
- Permette di avere dipendenze diverse per progetti diversi.
- **Evita conflitti** tra versioni di pacchetti.

## Installazione

- Aprire il terminale e digitare:

```
pip install virtualenv
```

- Crea un nuovo ambiente virtuale:

```
virtualenv -p python3 nome_ambiente
```

- Attiva l'ambiente virtuale:

- Windows

```
myenv\Scripts\activate
```

- Linux o Mac OS

```
source myenv/bin/activate
```

- Disattivare l'ambiente

```
deactivate
```

# Setup Python: Pip

## Cos'è pip?

- Sistema di **gestione dei pacchetti** per Python.
- Consente di **installare e gestire** librerie e dipendenze.

## Installazione

- Generalmente pre-installato con Python 3.4+.
- Se necessario: `python get-pip.py`

## Ricerca dei pacchetti

- È possibile cercare nel [PyPI store](#) per verificare se il pacchetto desiderato è **disponibile**.

## Comandi Principali

- **Installare un pacchetto:** `pip install nome_pacchetto`
- **Installare una versione specifica:** `pip install nome_pacchetto==versione`
- **Aggiornare un pacchetto:** `pip install --upgrade nome_pacchetto`
- **Disinstallare un pacchetto:** `pip uninstall nome_pacchetto`
- **Elencare i pacchetti installati:** `pip list`
- **Generare un file requirements:** `pip freeze > requirements.txt`
- **Installare da un file requirements:** `pip install -r requirements.txt`

# Jupyter Notebook

## Cos'è Jupyter Notebook?

- **Ambiente interattivo** per la programmazione in Python.
- Permette di combinare codice, testo, grafica e output in un unico documento.

## Caratteristiche Principali

- **Interattività:** Consente l'esecuzione di codice in tempo reale.
- **Markdown:** Supporta la formattazione del testo usando Markdown per documentazione e annotazioni.
- **Visualizzazione Dati:** Facilita la creazione di grafici e visualizzazioni direttamente all'interno del notebook.

## Installazione

- Installazione via pip:

```
pip install jupyter
```

- Avvio:

```
jupyter notebook
```

- Si apre un'interfaccia web per gestire i notebook.

## Esempio Interattivo

- È possibile provare Jupyter Notebook direttamente online attraverso il seguente [link](#)

# Hello World in Python, C e Java

## Python

```
print("Hello, World!")
```

## Linguaggio C

```
#include <stdio.h>

int main() {
    printf("Hello, World! \n");
    return 0;
}
```

## Java

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

## Considerazioni

- **Python:** Una singola riga di codice. Python è molto leggibile e richiede poche parole chiave.
- **C:** Necessità di includere librerie e scrivere una funzione **main()**. Serve anche gestire esplicitamente il ritorno dalla funzione.
- **Java:** Struttura più complessa, con classi e metodi richiesti, anche per un semplice output.

# Hello World in Python, C e Java

## Python (Interpretato)

- Viene eseguito direttamente dall'interprete, **senza una fase di compilazione**.
- **Esecuzione:** Scrivere il codice in un file .py ed eseguire

```
python hello.py
```

## C (Compilato)

- Il codice sorgente viene tradotto in codice macchina prima di essere eseguito.
- **Esecuzione:** Compilazione con gcc. Esecuzione dell'eseguibile generato.

```
gcc hello.c -o hello  
./hello
```

## Java (Compilazione + Interpretazione)

- Il codice sorgente viene compilato in bytecode.
- Il bytecode è eseguito dalla macchina virtuale Java (JVM).

```
javac HelloWorld.java  
java HelloWorld
```

# Python è un Linguaggio Tipizzato Dinamicamente

## Tipizzazione Dinamica

- In Python, non è necessario specificare esplicitamente il tipo delle variabili.
- Il tipo viene **dedotto automaticamente** dall'interprete in base al valore assegnato.

```
x = 10      # x è un intero  
y = "ciao"  # y è una stringa
```

## Assenza di punto e virgola

- In Python, non è necessario terminare ogni istruzione con il carattere ';' , a differenza di linguaggi come C e Java.

```
print("Hello, World!")  # Nessun punto e virgola richiesto
```

# Indentazione in Python

## L'Assenza del `;` e il Ruolo dell'Indentazione

- In Python, non è necessario utilizzare il punto e virgola (;) per terminare le istruzioni. L'interprete riconosce automaticamente la fine di un'istruzione grazie al **ritorno a capo**.

```
print("Hello, World!") # Nessun punto e virgola richiesto
```

## Uso dell'Indentazione:

- Invece di utilizzare parentesi graffe {} come in C o Java, Python si affida all'**indentazione** per definire i blocchi di codice.

```
if x > 0:  
    print("x è positivo") # Indentazione obbligatoria  
else:  
    print("x è negativo o zero")
```

- Ogni blocco di codice deve essere indentato con lo stesso numero di spazi. L'indentazione sbagliata genera un errore di sintassi:

```
if x > 0:  
    print("Blocco corretto")  
    print("Errore di indentazione") # Errore
```

# Tipi Primitivi in Python

## Tipi Primitivi

- **Interi** (int): Numeri interi, positivi o negativi.
- **Float** (float): Numeri decimali.
- **Stringhe** (str): Sequenze di caratteri.
- **Booleani** (bool): Valori logici, **True** o **False**.

## Esempio di Codice

```
# Nome della persona (Stringa)
nome = "Mario Rossi"

# Età della persona (Intero)
età = 30

# Altezza della persona in metri (Float)
altezza = 1.75

# Se la persona è sposata o no (Booleano)
sposato = False
```

# Formattazione di Stringhe e Numeri in Python

## Formattazione di Stringhe:

- Per rendere una **stringa più leggibile**, Python fornisce il metodo `.format()`.
- Questo metodo consente di inserire valori all'interno di una stringa **in posizioni specifiche**.
- Supporta anche la **formattazione dei numeri**, inclusi i numeri a virgola mobile.
- Per ulteriori dettagli: <https://pyformat.info/>

## Esempio di Codice

```
# Dichiarazione delle variabili
nome = "Mario"
eta = 30
altezza = 1.7534

# Formattazione di una stringa con il metodo .format()
info = "Il nome è {}, ha {} anni e un'altezza di {:.2f} metri.".format(nome, eta, altezza)

# Stampa della stringa formattata
print(info)
```

# Operazioni Matematiche in Python

## Operazioni Matematiche

- Python supporta le operazioni matematiche comuni in modo intuitivo e diretto.
- Le operazioni possono essere eseguite su numeri interi (**int**) e numeri a virgola mobile (**float**).

## Esempio di Codice

```
# Dichiarazione delle variabili
a = 10          # Intero
b = 3           # Intero
c = 2.5         # Float

# Operazioni matematiche
somma = a + b            # Somma
differenza = a - b       # Sottrazione
prodotto = a * b          # Moltiplicazione
quoziente = a / b         # Divisione (risultato float)
intero_divisione = a // b # Divisione intera
resto = a % b             # Resto della divisione
esponente = a ** c        # Elevamento a potenza
```

# Caratteristiche delle Stringhe in Python

## Stringhe come Sequenze di Caratteri

- Le stringhe sono sequenze ordinate di caratteri racchiuse tra virgolette singole ('') o doppie ("").
- Immutabilità:** Le stringhe non supportano l'assegnazione di elementi. Una volta creata, non è possibile modificare un singolo carattere.

```
parola = "Python"  
# parola[0] = "J" # Questo causerà un errore: TypeError
```

- Tipo di Dato:** Le stringhe non possono essere mescolate direttamente con tipi numerici come `int` o `float`.

```
numero = 10  
# messaggio = "Il numero è " + numero # Questo causerà un errore: TypeError
```

- Correzione dell'Errore:** Per concatenare una stringa con un numero, è necessario convertire il numero in una stringa utilizzando `str()`.

```
messaggio = "Il numero è " + str(numero) # Ora funziona correttamente  
print(messaggio)
```

# Collezioni in Python – Le Tuple

## Collezioni in Python

- Python offre strutture dati più complesse per gestire insiemi di valori.
- Una di queste strutture sono le **tuple**.

## Tuple

- Le **tuple** sono collezioni **immutabili** di valori separati da virgole.
- Possono contenere valori di qualsiasi tipo (interi, stringhe, float, ecc.).
- I valori all'interno di una tupla possono essere **accessibili tramite l'indice**, ma non possono essere modificati.

```
# Creazione di una tupla
informazioni = ("Mario", 30, 1.75)

# Accesso ai valori tramite indice
print(informazioni[0])    # Stampa 'Mario'
print(informazioni[1])    # Stampa 30
print(informazioni[0:2])   # Output: ('Mario', 30)

# Tentativo di modifica del valore (causa un errore)
# informazioni[1] = 31 # Questo causerà un TypeError
```

# Le Liste in Python – Collezioni Mutabili

## Liste: Collezioni Mutabili

- Per raccogliere dati mutabili, una scelta migliore rispetto alle tuple è utilizzare le liste.
- Le liste, a differenza delle tuple, possono essere modificate dopo la loro creazione.

## Accesso e Modifica degli Elementi

- Come per le tuple, è possibile accedere agli elementi di una lista tramite il loro indice.
- Tuttavia, diversamente dalle tuple, le liste sono modificabili: possiamo aggiungere, rimuovere o cambiare elementi.

## Metodi Principali

- **append()**: Aggiunge un elemento alla fine della lista.
- **remove()**: Rimuove un elemento specifico dalla lista.
- **insert(index, elem)**: Inserisce un elemento in una posizione specifica.
- **pop()**: Rimuove e restituisce l'ultimo elemento (o l'elemento in un indice specifico).
- **clear()**: Rimuove tutti gli elementi dalla lista.
- **sort()**: Ordina gli elementi della lista (se applicabile).

# Le Liste in Python – Collezioni Mutabili

## Esempio di Codice

```
# Creazione di una lista
informazioni = ["Mario", 30, 1.75]

# Accesso agli elementi tramite indice
print(informazioni[0])    # Output: Mario

# Modifica di un elemento (modifica età)
informazioni[1] = 31      # Output: ['Mario', 31, 1.75]

# Aggiunta di un nuovo elemento alla fine (append)
informazioni.append("Ingegnere")  # Output: ['Mario', 31, 1.75, 'Ingegnere']

# Inserimento di un elemento in una posizione specifica (insert)
informazioni.insert(1, "Rossi")  # Output: ['Mario', 'Rossi', 31, 1.75, 'Ingegnere']

# Rimozione di un elemento specifico (remove)
informazioni.remove(1.75) # Output: ['Mario', 'Rossi', 31, 'Ingegnere']

# Rimozione e ritorno dell'ultimo elemento (pop)
informazioni.pop() # Output: ['Mario', 'Rossi', 31]

# Aggiunta di numeri per l'ordinamento e ordinamento (sort)
informazioni.extend([5, 10, 2])
informazioni.sort()  # Output: [2, 5, 10, 'Mario', 'Rossi']

# Rimozione di tutti gli elementi (clear)
informazioni.clear()  # Output: []
```

# I Dizionari in Python – Coppie Chiave-Valore

## Dizionari: Collezioni di Coppie Chiave-Valore

- I **dizionari** in Python sono collezioni non ordinate che permettono di memorizzare dati sotto forma di **coppie chiave-valore**.
- Ogni chiave è **unica** e viene utilizzata per accedere al valore associato.

## Caratteristiche dei Dizionari

- **Chiavi:** Devono essere di un tipo immutabile (ad esempio, stringhe, numeri, tuple).
- **Valori:** Possono essere di qualsiasi tipo, incluse altre liste o dizionari.
- **Mutabilità:** I valori possono essere modificati, aggiunti o rimossi, ma le chiavi non possono essere cambiate una volta assegnate.

## Metodi Utili per i Dizionari

- **get(key)**: Restituisce il valore associato a una chiave.
- **keys()**: Restituisce tutte le chiavi del dizionario.
- **values()**: Restituisce tutti i valori del dizionario.
- **items()**: Restituisce tutte le coppie chiave-valore come tuple.
- **update()**: Aggiorna il dizionario con coppie chiave-valore da un altro dizionario.

# Le Liste in Python – Collezioni Mutabili

## Esempio di Codice

```
# Creazione di un dizionario
persona = {
    "nome": "Mario",
    "età": 30,
    "professione": "Ingegnere"
}

# Accesso ai valori tramite chiavi
print(persona["nome"])    # Output: Mario

# Aggiunta di una nuova coppia chiave-valore
persona["altezza"] = 1.75  # Output: {'nome': 'Mario', 'età': 30, 'professione': 'Ingegnere',
                           'altezza': 1.75}

# Modifica del valore associato a una chiave
persona["età"] = 31       # Output: {'nome': 'Mario', 'età': 31, 'professione': 'Ingegnere',
                           'altezza': 1.75}

# Rimozione di una coppia chiave-valore
del persona["professione"] # Output: {'nome': 'Mario', 'età': 31, 'altezza': 1.75}
```

# Statements in Python – Condizioni e Cicli

## Statements in Python

- Come in altri linguaggi di programmazione, Python offre istruzioni come **if** per le condizioni e **for/while** per le iterazioni.

```
# Esempio con if, elif, else
età = 17

if età >= 18:
    print("Sei maggiorenne")
elif età >= 16:
    print("Puoi guidare con il permesso")
else:
    print("Sei minorenne")
```

# Il Ciclo For in Python – Iterazioni Semplici e su Collezioni

## Ciclo for

- Il ciclo for permette di eseguire un insieme di istruzioni per un numero specifico di volte (N).
- È particolarmente utile per iterare su collezioni come liste, tuple, dizionari, set e stringhe.

## Sintassi del ciclo for

```
# Esecuzione di un'istruzione N volte
for i in range(inizio, fine, passo):  # Itera da 'inizio' a 'fine - 1' con incrementi di 'passo'
    print("Iterazione:", i)
```

- In questo esempio, range(inizio, fine, passo) consente di specificare:
  - **Inizio**: valore iniziale (incluso)
  - **Fine**: valore finale (escluso)
  - **Passo**: incremento tra i valori (default è 1)

## Esempio di Ciclo For con Step

```
# Iterazione con passo
for i in range(0, 10, 2):  # Itera da 0 a 8, con passo di 2
    print("Iterazione:", i)
```

# Il Ciclo For in Python – Iterazioni Semplici e su Collezioni

## Iterazione su Collezioni

- È possibile utilizzare il ciclo for per iterare direttamente su collezioni, ad esempio liste o dizionari.

### Esempio di Iterazione su una Lista

```
# Iterazione su una lista di nomi
nomi = ["Mario", "Luca", "Anna"]

for nome in nomi:
    print("Ciao", nome)
```

### Esempio di Iterazione su un Dizionario

```
# Iterazione su un dizionario
persona = {"nome": "Mario", "età": 30}

for chiave, valore in persona.items():
    print(chiave + ":", valore)
```

# Funzioni in Python

## Funzioni

- Per migliorare sia la **leggibilità** che la **riutilizzabilità** del codice, è buona pratica encapsulare un insieme di istruzioni all'interno di una funzione.
- Le funzioni possono essere chiamate sia dallo stesso script che da altri script.
- Le funzioni in Python vengono definite utilizzando la parola chiave `def`.

## Esempio di Funzione

```
# Definizione di una funzione per sommare due numeri
def somma(x, y):
    risultato = x + y
    return risultato

# Chiamata alla funzione
res = somma(5, 3)
print("La somma è:", res)
```

- In questo esempio, `x` e `y` sono gli **argomenti** della funzione `somma`.

# Valori di Default nelle Funzioni in Python

## Valori di Default

- In Python, è possibile assegnare **valori di default** agli argomenti di una funzione (codice flessibile e gestibile)
- Questo consente di chiamare la funzione senza specificare tutti gli argomenti, utilizzando i valori predefiniti se necessario.

## Esempio di Funzione con Valori di Default

```
# Funzione con valore di default
def saluto(nome, messaggio="Ciao!"):
    print(messaggio, nome)

# Chiamate alla funzione
saluto("Mario")           # Usa il messaggio di default
saluto("Luca", "Benvenuto!") # Usa un messaggio specifico
```

- In questo esempio, se non viene fornito un messaggio, verrà utilizzato "Ciao!" come valore predefinito.
- **Modifica di Argomenti:** È possibile modificare un insieme specifico di valori quando si chiama la funzione, senza alterare i valori predefiniti.

# Valori di Default nelle Funzioni in Python

## Valori di Default

- In Python, è possibile assegnare **valori di default** agli argomenti di una funzione.
- Questo consente di chiamare la funzione senza specificare tutti gli argomenti, utilizzando i valori predefiniti se necessario.

## Attenzione all'Ordine

- **L'ordine degli argomenti è cruciale:** gli argomenti senza valori di default devono precedere quelli con valori di default nella definizione della funzione.

```
# Questo causerà un errore di sintassi
def esempio(arg1="Default", arg2):
    pass # Non fare nulla
```

- Questo codice genererà un errore, poiché arg2 (senza valore di default) segue arg1 (con valore di default).

# Scope delle Variabili in Python – Visibilità e Accessibilità

## Definizione di Scope

- Lo **scope** di una variabile definisce la sua **visibilità** e accessibilità nel codice.
- Dipende da dove la variabile è stata definita e influisce su come possiamo utilizzare quella variabile.

## Dichiarazione delle Variabili

- In Python, non è sempre necessario dichiarare esplicitamente una variabile. Tuttavia, è considerata una buona pratica farlo per migliorare la **leggibilità** del codice.

## Esempio di Scope

```
x = 5 # Variabile definita a livello globale

if x > 3:
    b = True # 'b' viene definita solo se la condizione è vera

print(b) # Stampa True se x > 3
```

- In questo caso, **b** viene definita se la condizione **x > 3** è verificata. Se **x** è maggiore di 3, **b** avrà il valore **True**.

# Scope delle Variabili in Python – Visibilità e Accessibilità

## Definizione di Scope

- Lo **scope** di una variabile definisce la sua **visibilità** e accessibilità nel codice.
- Dipende da dove la variabile è stata definita e influisce su come possiamo utilizzare quella variabile.

## Dichiarazione delle Variabili

- In Python, non è sempre necessario dichiarare esplicitamente una variabile. Tuttavia, è considerata una buona pratica farlo per migliorare la **leggibilità** del codice.

## Esempio di Scope (Possibile Errore)

```
x = 1 # Variabile definita a livello globale

if x > 3:
    b = True # 'b' non viene definita

print(b) # Questo causerà un errore: NameError: name 'b' is not defined
```

- Con `x = 1`, il programma genererà un errore `NameError` poiché `b` non è stata definita.

# Tipi di Scope in Python

## Scope delle Variabili

- Lo scope di una variabile determina la sua **visibilità** e **accessibilità** nel codice.
- In Python, ci sono tre principali tipi di scope: **globale**, **locale**, **non locale**

### 1. Globale

- Variabili definite al di fuori di funzioni o classi.
- Accessibili in tutto il programma, inclusi i blocchi di codice e le funzioni.

#### Esempio

```
x = 10 # Variabile globale

def stampa_x():
    print(x) # Può accedere alla variabile globale

stampa_x() # Output: 10
```

# Tipi di Scope in Python

## Scope delle Variabili

- Lo scope di una variabile determina la sua **visibilità e accessibilità** nel codice.
- In Python, ci sono tre principali tipi di scope: **globale, locale, non locale**

## 2. Locale

- Variabili definite all'interno di funzioni o blocchi di codice.
- Accessibili solo all'interno di quel contesto.

### Esempio

```
def funzione_locale():
    y = 5    # Variabile locale
    print(y)  # Accesso consentito

funzione_locale()  # Output: 5
# print(y)  # Questo causerà un errore: NameError: name 'y' is not defined
```

# Tipi di Scope in Python

## Scope delle Variabili

- Lo scope di una variabile determina la sua **visibilità e accessibilità** nel codice.
- In Python, ci sono tre principali tipi di scope: **globale, locale, non locale**

### 3. Non Locale

- Variabili definite in un contesto di funzioni annidate.
- Accessibili nella funzione interna, ma non globali.

#### Esempio

```
def funzione_non_locale():
    z = 15 # Variabile locale nella funzione esterna

    def funzione_interna():
        print(z) # Può accedere alla variabile non locale

    funzione_interna() # Output: 15

funzione_non_locale()
```

# Oggetti in Programmazione Orientata agli Oggetti (OOP)

## Definizione di Oggetto

- In OOP, un **oggetto** incapsula **dati** e **metodi** per gestire un elemento specifico del dominio.
- Gli oggetti sono **istanze** di una classe, che definisce il modello per questi oggetti.

## Esempio

- Immaginiamo di voler modellare una persona. La classe **Persona** può contenere tutte le informazioni rilevanti (nome, età, professione) e metodi per interagire con queste informazioni.

```
# Definizione della classe Persona
class Persona:
    def __init__(self, nome, età, professione):
        self.nome = nome
        self.età = età
        self.professione = professione

# Creazione di un oggetto Mario
mario = Persona(nome="Mario Rossi", età=30, professione="Ingegnere")
```

# Oggetti in Programmazione Orientata agli Oggetti (OOP)

## Definizione di Oggetto

- In OOP, un **oggetto** incapsula **dati** e **metodi** per gestire un elemento specifico del dominio.
- Gli oggetti sono **istanze** di una classe, che definisce il modello per questi oggetti.

## Dati e Metodi

- Un oggetto come **mario** ha le sue **proprietà** (nome, età, professione) e può avere **metodi** che eseguono operazioni su questi dati.

```
# Metodo che restituisce una descrizione della persona
def descrizione(self):
    return f"{self.nome}, {self.età} anni, è un {self.professione}"

# Chiamata del metodo per Mario
print(mario.descrizione()) # Output: Mario Rossi, 30 anni, è un Ingegnere
```

# Costruttore ed Ereditarietà in Python

## Ereditarietà in Python

- In Python, tutte le classi **ereditano** automaticamente dalla classe base **object**.
- Questo vale anche per la nostra classe **Persona**, che eredita implicitamente metodi e proprietà dalla classe **object**.

## Definizione del Costruttore

- Il **costruttore** viene definito con il metodo speciale **\_\_init\_\_**.
- Il costruttore inizializza gli attributi della classe, come il **nome** e l'**ID** per l'oggetto **Persona**.

```
class Persona:  
    def __init__(self, nome, id):  
        self.nome = nome # Attributo nome  
        self.id = id      # Attributo ID
```

# Costruttore ed Ereditarietà in Python

## Ereditarietà in Python

- In Python, tutte le classi **ereditano** automaticamente dalla classe base **object**.
- Questo vale anche per la nostra classe **Persona**, che eredita implicitamente metodi e proprietà dalla classe **object**.

## La Parola Chiave self

- **self** funziona come **this** in Java: si riferisce all'oggetto corrente.
- Permette di accedere e modificare gli attributi dell'oggetto all'interno dei metodi della classe.

```
class Persona:  
    def __init__(self, nome, id):  
        self.nome = nome # Assegna l'attributo nome  
        self.id = id      # Assegna l'attributo ID  
  
    # Creazione dell'oggetto Mario  
mario = Persona("Mario Rossi", "0123456")
```

- Quando creiamo un'istanza della classe **Persona**, il costruttore **\_\_init\_\_** assegna i valori agli attributi.

# Costruttori e Setters in Python

## Costruttori in Python

- In Python **non è possibile** definire **più costruttori** come in altri linguaggi (es. Java).
- Tuttavia, possiamo utilizzare **valori predefiniti** per gli argomenti del costruttore per ottenere un comportamento simile.

## Esempio con Valori Predefiniti

- Definiamo un costruttore con valori di default per gestire diversi scenari di inizializzazione.

```
class Persona:  
    def __init__(self, nome="Mario Rossi", id="000000", età=30):  
        self.nome = nome  
        self.id = id  
        self.__età = età # Attributo privato
```

- Se non vengono forniti valori, il costruttore utilizzerà quelli **predefiniti** (nome="Mario Rossi", id="000000", età=30).

# Costruttori e Setters in Python

## Setters per Attributi Privati

- In Python, possiamo definire **setters** per modificare attributi privati, utilizzando una convenzione (es. `__età`) per indicare la **privacy** dell'attributo.
- I **setters** permettono di gestire le modifiche agli attributi privati dall'esterno della classe.

```
class Persona:  
    def __init__(self, nome, id, età=30):  
        self.nome = nome  
        self.id = id  
        self.__età = età # Attributo privato  
  
    # Setter per l'età  
    def set_età(self, nuova_eta):  
        if nuova_eta > 0:  
            self.__età = nuova_eta  
  
    # Getter per accedere all'età  
    def get_età(self):  
        return self.__età
```

# Costruttori e Setters in Python

## Utilizzo dei Setters

- Possiamo usare i setters per cambiare valori di attributi privati in modo controllato.

```
mario = Persona("Mario Rossi", "0123456")
print(mario.get_eta()) # Output: 30
```

```
mario.set_eta(35)
print(mario.get_eta()) # Output: 35
```

# Ereditarietà in Python

## Ereditarietà

- In Python, è possibile **estendere** le classi per **riutilizzare** metodi e proprietà della classe genitore (parent).
- L'ereditarietà ci consente di creare classi più specifiche, basate su classi più generiche, aggiungendo o modificando funzionalità.

```
class Persona:  
    def __init__(self, nome, età):  
        self.nome = nome  
        self.età = età  
  
    def descrizione(self):  
        return f"{self.nome}, {self.età} anni"  
  
# La classe Studente eredita da Persona  
class Studente(Persona):  
    def __init__(self, nome, età, id_studente):  
        super().__init__(nome, età) # Richiama il costruttore del genitore  
        self.id_studente = id_studente  
  
    def descrizione_completa(self):  
        return f"{self.descrizione()}, ID studente: {self.id_studente}"
```

- La parola chiave **super()** permette di richiamare il costruttore e i metodi della classe genitore.

# Ereditarietà in Python

## Esempio di Utilizzo

```
mario = Studente("Mario Rossi", 30, "0123456")
print(mario.descrizione_completa()) # Output: Mario Rossi, 30 anni, ID studente: 0123456
```

- `mario` è un'istanza di **Studente**, ma può accedere ai metodi e agli attributi definiti nella classe **Persona**.

## Ereditarietà Multipla

- Python supporta anche l'ereditarietà multipla, permettendo a una classe di ereditare da più classi.

```
class Sportivo:
    def pratica_sport(self):
        return "Pratica uno sport"

class StudenteSportivo(Studente, Sportivo):
    pass

mario_sportivo = StudenteSportivo("Mario Rossi", 30, "0123456")
print(mario_sportivo.pratica_sport()) # Output: Pratica uno sport
```

- **StudenteSportivo** eredita da **Studente** e **Sportivo**, combinando metodi e proprietà da entrambe le classi.

# Imports in Python

## Imports

- A volte, è necessario **importare funzioni o oggetti** definiti in altri file o moduli.
- Python ci permette di farlo usando la parola chiave **import**.

## Ereditarietà Multipla

```
# Import di un modulo creato dall'utente (utils.py)
import utils

# Import di una funzione specifica dal modulo
from utils import somma

# Import di una libreria di terze parti
import numpy as np
```

## Installazione di Librerie di Terze Parti

- Le **librerie di terze parti** come **NumPy**, **TensorFlow**, ecc., devono essere **installate** nel sistema o nell'ambiente virtuale.
- Per farlo, è possibile utilizzare un gestore di pacchetti come **pip** o **Anaconda**.

# **Libreria Numpy: Una Guida Rapida**

# Introduzione a NumPy



## Cos'è NumPy? ([Documentazione Online](#))

- **NumPy (Numerical Python)** è un pacchetto di calcolo scientifico open-source per Python.
- È progettato per fornire un potente supporto per la manipolazione di **array N-dimensionali** e per l'esecuzione di **operazioni matematiche** su di essi.

## Caratteristiche Principali

- **Array N-Dimensionali:** Un potente oggetto array N-dimensionale (**ndarray**) che consente di gestire dati complessi in modo efficiente.
- **Funzioni Avanzate:** Include funzioni sofisticate per il broadcasting, che permettono operazioni su array di forme diverse in modo intuitivo.
- **Integrazione con C/C++ e Fortran:** Strumenti per integrare codice scritto in C/C++ e Fortran.
- **Capacità di Algebra Lineare e Statistiche:** Fornisce funzionalità utili per l'algebra lineare, la trasformata di Fourier e la generazione di numeri casuali.

## Utilizzo di NumPy

- NumPy è ampiamente utilizzato in molte librerie scientifiche e di machine learning, tra cui:
  - **OpenCV:** Per l'elaborazione delle immagini.
  - **TensorFlow e PyTorch:** Per l'addestramento e l'esecuzione di modelli di deep learning.

# Come Installare NumPy

## Installazione di NumPy

- NumPy può essere facilmente installato utilizzando **pip**, il gestore di pacchetti per Python.
- Aprire il terminale e digitare

```
pip install numpy
```

## Importare NumPy nel Proprio Script

- Dopo l'installazione, è possibile importare NumPy nel proprio script Python con il seguente comando:

```
import numpy as np
```

# Array N-Dimensionali in NumPy

## Cosa sono gli Array N-Dimensionali?

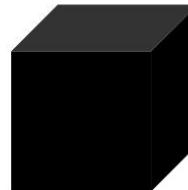
- NumPy supporta array N-dimensionali, che consentono di gestire dati complessi in più dimensioni.
- Gli array possono avere una forma (shape) che definisce il numero di dimensioni e il numero di elementi in ciascuna dimensione.



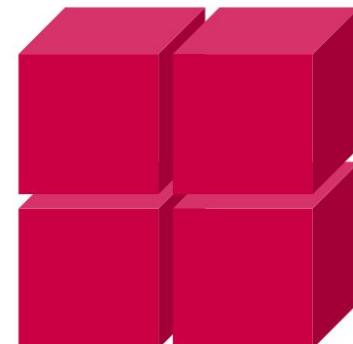
1D



2D



3D



4D

# Creare Array in NumPy

## Creazione di un Array

- Per creare un array in NumPy, si può utilizzare la funzione `array`, passando una lista di valori e, facoltativamente, il tipo di dati desiderato.

## Esempio di Creazione di un Array

```
import numpy as np

# Creazione di un array con valori interi
array_int = np.array([1, 2, 3, 4, 5])

# Creazione di un array con valori di tipo float
array_float = np.array([1.0, 2.0, 3.0, 4.0, 5.0], dtype=float)

# Creazione di un array di stringhe
array_str = np.array(['a', 'b', 'c'], dtype=str)
```

## Note

- Array Omogenei:** Gli array NumPy devono essere omogenei, il che significa che tutti gli elementi devono avere lo stesso tipo di dato.
- Tipo di Dato Predefinito:** Se il tipo di dato non viene specificato, NumPy imposterà il tipo di dato su `float64` per impostazione predefinita.

# Esempi di Array Multidimensionali in NumPy

## Array 1-Dimensionale (Vettore)

- Un array con una singola dimensione.

```
import numpy as np

array_1d = np.array([1, 2, 3, 4, 5])
```

## Array 2-Dimensionale (Matrice)

- Un array con due dimensioni, utile per rappresentare matrici.

```
array_2d = np.array([[1, 2, 3], [4, 5, 6]])
```

## Array 3-Dimensionale (Tensore)

- Un array con tre dimensioni, utile per rappresentare dati più complessi, come volumi o immagini RGB.

```
array_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

# Funzioni Standard per Creare Array in NumPy

## Funzioni per Creare Array

- NumPy offre diverse funzioni standard per creare array in modo semplice e veloce. Alcune di queste sono:
  - **ones**: Crea un array pieno di 1.
  - **zeros**: Crea un array pieno di 0.
  - **ones\_like** e **zeros\_like**: Creano array pieni di 1 o 0 con la stessa forma di un array esistente.
  - **eye**: Crea una matrice identità.

```
# Creare un array 3x3 pieno di zeri
array_zeros = np.zeros((3, 3))

# Creare un array 2x4 pieno di uno
array_ones = np.ones((2, 4))

# Creare una matrice identità 4x4
identity_matrix = np.eye(4)

# Creare un array pieno di uno con la stessa forma di un altro array
existing_array = np.array([[5, 6], [7, 8]])
array_ones_like = np.ones_like(existing_array)

# Creare un array pieno di zeri con la stessa forma di un altro array
array_zeros_like = np.zeros_like(existing_array)
```

# Creazione di Array con arange in NumPy

## Funzione `arange(start, stop, step)`

- La funzione `arange` genera un array contenente una sequenza di valori.
- È possibile specificare un valore di partenza (`start`), un valore di fine (`stop`), e il passo (`step`) tra gli elementi consecutivi.

## Caratteristiche

- Gli elementi generati appartengono all'insieme **semiaperto** [start, stop[, quindi `stop` **non è incluso**.
- Utilizzata spesso per creare array con valori regolari.

```
# Array da 0 a 9 (stop non incluso)
array1 = np.arange(0, 10)
```

```
# Array da 1 a 10 con step 2 (stop non incluso)
array2 = np.arange(1, 10, 2)
```

```
# Array con numeri negativi, da -10 a -2 con step 2
array3 = np.arange(-10, -2, 2)
```

# Generazione di Valori Casuali in NumPy

## Generazione di Valori Casuali con `rand`:

- La funzione `rand` di NumPy viene utilizzata per creare un array riempito con valori casuali.
- I valori generati seguono una distribuzione uniforme tra 0 e 1.

## Esempio

```
import numpy as np

# Creare un array 3x3 di valori casuali
random_array = np.random.rand(3, 3)

print("Array 3x3 di valori casuali:\n", random_array)

# Output
Array 3x3 di valori casuali:
[[0.54985767 0.87119432 0.73533241]
 [0.10284756 0.65021342 0.32977436]
 [0.95320177 0.47473821 0.83261454]]
```

# Creazione di Array con un Valore Scalare in NumPy

## Funzione `full`:

- La funzione `full` permette di creare un array di una forma specifica, riempito con un **singolo valore scalare**.

## Esempio con `full`

```
# Creare un array 3x3 riempito con il valore 7
scalar_array = np.full((3, 3), 7)

# Output
[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

## Alternative

- È possibile ottenere lo stesso risultato con `ones` o `ones_like`, moltiplicando il risultato per il valore desiderato.

```
# Alternativa con ones
scalar_array_alt = np.ones((3, 3)) * 7
```

# Attributi degli Array in NumPy

## Attributi degli Array

- Ogni array in NumPy ha attributi che forniscono informazioni importanti sull'array stesso.

## Esempio

- **dtype**: Restituisce il tipo dei dati contenuti nell'array.
- **shape**: Indica la dimensione dell'array lungo ogni dimensione.

```
# Creare un array 3x2 di float
arr = np.array([[1.5, 2.3], [3.1, 4.8], [5.6, 6.9]])

# Ottenere dtype e shape
print("Tipo dei dati:", arr.dtype)
print("Dimensioni dell'array:", arr.shape)

# Output
Tipo dei dati: float64
Dimensioni dell'array: (3, 2)
```

# Gestione delle Dimensioni degli Array in NumPy

## Aggiungere e Rimuovere Dimensioni

- **expand\_dims**: Aggiunge una nuova dimensione a un array esistente.
- **squeeze**: Rimuove tutte le voci di dimensione singola (1) da un array.

## Esempio di `expand_dims`

```
# Creare un array 3D utilizzando np.full
x = np.full((2, 2, 3), 7)

# Aggiungere una nuova dimensione in diverse posizioni
expanded_arr_axis_0 = np.expand_dims(x, axis=0)    # Aggiunge lungo l'asse 0
expanded_arr_axis_1 = np.expand_dims(x, axis=1)    # Aggiunge lungo l'asse 1
expanded_arr_axis_neg1 = np.expand_dims(x, axis=-1)  # Aggiunge lungo l'ultimo asse

# Verifiche delle forme
assert expanded_arr_axis_0.shape == (1, 2, 2, 3)    # Verifica forma dopo l'espansione sull'asse 0
assert expanded_arr_axis_1.shape == (2, 1, 2, 3)    # Verifica forma dopo l'espansione sull'asse 1
assert expanded_arr_axis_neg1.shape == (2, 2, 3, 1)  # Verifica forma dopo l'espansione sull'asse -1
```

# Riformattazione degli Array in NumPy

## Riformattare un Array

- Una delle operazioni comuni consiste nel cambiare la forma di un array esistente.
- Ad esempio, è possibile trasformare un array con 10 elementi in una matrice 2x5 utilizzando la funzione reshape.

## Utilizzo di reshape

- **Sintassi:** `array.reshape(new_shape)`
- È possibile utilizzare -1 in uno dei parametri di forma per consentire a NumPy di calcolare automaticamente la dimensione mancante.

```
# Creare un array 1D con 10 elementi
arr = np.arange(10) # Crea un array: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Riformattare l'array in una matrice 2x5
reshaped_arr = arr.reshape(2, 5)

# Riformattare utilizzando -1 per calcolare automaticamente una dimensione
reshaped_arr_auto = arr.reshape(2, -1) # NumPy calcola automaticamente la seconda dimensione

# Verifiche delle forme
assert reshaped_arr.shape == (2, 5) # Verifica forma dopo la riformattazione
assert reshaped_arr_auto.shape == (2, 5) # Verifica forma dopo la riformattazione con -1
```

# Rifommattazione degli Array in NumPy

## Condizioni per l'Uso di reshape

- Una Dimensione Sconosciuta:** È possibile utilizzare `-1` per indicare che NumPy deve calcolare automaticamente una dimensione, ma solo se è presente una dimensione sconosciuta.
- Più Dimensioni Sconosciute:** Se si tenta di utilizzare `-1` per più dimensioni sconosciute, NumPy genererà un errore di tipo `ValueError`.

```
# Creare un array 1D con 12 elementi
arr = np.arange(12) # Crea un array: [0, 1, 2, ..., 11]

# Rifommattare in una matrice 3x4
reshaped_arr = arr.reshape(3, 4) # Funziona senza problemi

# Rifommattare utilizzando -1
reshaped_arr_auto = arr.reshape(3, -1) # NumPy calcola automaticamente la seconda dimensione

# Tentativo di rifommattare con più dimensioni sconosciute
try:
    invalid_reshape = arr.reshape(-1, -1) # Questo solleverà un errore
except ValueError as e:
    print("Errore:", e) # Stampa l'errore
```

# Accesso agli Elementi di un Array in NumPy

## Accesso tramite Notazione per Indice

- In NumPy, gli elementi di un array possono essere acceduti utilizzando la notazione per indice.
- Gli indici partono da 0, quindi il primo elemento ha indice 0.

```
# Creare un array 1D
arr = np.array([10, 20, 30, 40, 50])

# Accesso agli elementi tramite notazione per indice
element_0 = arr[0] # Primo elemento
element_2 = arr[2] # Terzo elemento
```

## Accesso tramite item

- Gli elementi possono essere recuperati anche utilizzando la funzione `item()`, che restituisce il valore dell'elemento in una posizione specifica.

```
# Accesso agli elementi usando la funzione item
element_1 = arr.item(1) # Secondo elemento
```

# Notazione di Slicing in NumPy

## Introduzione allo Slicing

- La notazione di slicing, utilizzata anche per le stringhe in Python, è valida anche per gli array NumPy. Consente di accedere a una porzione dell'array in modo semplice e intuitivo.

## Sintassi di Slicing

- La sintassi di slicing è la seguente: `array[start:stop:step]`
  - **start:** Indice di inizio (incluso)
  - **stop:** Indice di fine (escluso)
  - **step:** Incremento (opzionale)

## Esempio di Slicing

```
# Creare un array 1D
arr = np.array([10, 20, 30, 40, 50, 60, 70, 80])

# Slicing dell'array
slice_1 = arr[2:5] # Elementi dall'indice 2 all'indice 4
slice_2 = arr[::2] # Ogni secondo elemento
```

# Concatenazione di Array in NumPy

## Concatenazione

- In NumPy, è possibile concatenare due o più array per ottenere un singolo array come output.
- La funzione `concatenate` consente di combinare array lungo un determinato asse.

## Sintassi di `concatenate`

- La sintassi di slicing è la seguente: `numpy.concatenate((array1, array2, ...), axis=0)`
  - **array1, array2, ...:** gli array da concatenare.
  - **axis:** l'asse lungo il quale concatenare. (0 per righe, 1 per colonne)

## Esempio di Concatenazione

```
# Creare due array
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Concatenare gli array
concatenated_array = np.concatenate((arr1, arr2))

print("Array Concatenato:", concatenated_array) # Stampa: [1 2 3 4 5 6]
```

# Operazioni Matematiche con NumPy

## Numpy Math

- NumPy offre una vasta gamma di funzioni matematiche, dalle operazioni di base a quelle più complesse, applicabili direttamente sugli array.

## Operazioni Matematiche di Base

- Permette di eseguire facilmente operazioni matematiche su array interi o float, come somma, sottrazione, moltiplicazione e divisione.

```
import numpy as np

# Creare un array
arr = np.array([1, 2, 3, 4])

# Operazioni matematiche
arr_sum = np.sum(arr)          # Somma degli elementi: 10
arr_prod = np.prod(arr)         # Prodotto degli elementi: 24
arr_mean = np.mean(arr)         # Media degli elementi: 2.5
```

# Operazioni Matematiche con NumPy

## Numpy Math

- NumPy offre una vasta gamma di funzioni matematiche, dalle operazioni di base a quelle più complesse, applicabili direttamente sugli array.

## Operazioni Element-wise

- Le operazioni matematiche possono essere eseguite elemento per elemento.

```
# Creare due array
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Sommare gli array
arr_sum = np.add(arr1, arr2)    # [5, 7, 9]

# Moltiplicare gli array
arr_mult = np.multiply(arr1, arr2) # [4, 10, 18]
```

# Limiti del Broadcasting in NumPy

## Condizioni per il Broadcasting

- Due array sono **compatibili per il broadcasting** se:
  - Hanno lo stesso numero di dimensioni oppure
  - Differiscono per una sola dimensione, dove uno degli array ha una dimensione pari a 1.
- Il risultato finale avrà la dimensione massima per ogni dimensione corrispondente.

```
import numpy as np

# Array di shape (4, 2, 3)
x = np.ones((4, 2, 3))

# Array incompatibile per broadcasting (shape (2, 3, 3))
y = np.ones((2, 3, 3))

# Questo solleverà ValueError
try:
    array_sum = x + y
except ValueError as e:
    print("Errore:", e)
```

# Limiti del Broadcasting in NumPy

## Condizioni per il Broadcasting

- Due array sono **compatibili per il broadcasting** se:
  - Hanno lo stesso numero di dimensioni oppure
  - Differiscono per una sola dimensione, dove uno degli array ha una dimensione pari a 1.
- Il risultato finale avrà la dimensione massima per ogni dimensione corrispondente.

```
import numpy as np

# Array di shape (4, 1, 3)
x = np.ones((4, 1, 3))

# Array di shape (4, 2, 3)
y = np.ones((4, 2, 3))

# Broadcasting applicato correttamente
array_sum = x + y
```

# Moltiplicazione Element-wise in NumPy

## Moltiplicazione con \* o np.multiply()

- In NumPy, possiamo eseguire la moltiplicazione elemento per elemento tra due array usando l'operatore \* o la funzione np.multiply().

```
import numpy as np

# Array 1: shape (2, 3)
x = np.array([[1, 2, 3], [4, 5, 6]])

# Array 2: shape (2, 3)
y = np.array([[7, 8, 9], [10, 11, 12]])

# Moltiplicazione elemento per elemento con *
result_star = x * y

# Moltiplicazione elemento per elemento con np.multiply()
result_multiply = np.multiply(x, y)
```

# Moltiplicazione di Matrici in NumPy

## Moltiplicazione di Matrici con `np.matmul()`

- La moltiplicazione di matrici può essere effettuata utilizzando la funzione `np.matmul()`
- Questa funzione è adatta sia per matrici bidimensionali che per matrici con dimensioni superiori a due.

### Esempio 1: Moltiplicazione di Matrici 2D

```
import numpy as np

# Matrice 1: shape (2, 3)
A = np.array([[1, 2, 3], [4, 5, 6]])

# Matrice 2: shape (3, 2)
B = np.array([[7, 8], [9, 10], [11, 12]])

# Moltiplicazione di matrici 2D
result_2d = np.matmul(A, B)
```

# Moltiplicazione di Matrici in NumPy

## Moltiplicazione di Matrici con `np.matmul()`

- La moltiplicazione di matrici può essere effettuata utilizzando la funzione `np.matmul()`
- Questa funzione è adatta sia per matrici bidimensionali che per matrici con dimensioni superiori a due.

## Esempio 2: Moltiplicazione di Matrici N-Dimensionali con Differenti Dimensioni

```
import numpy as np

# Array 1: forma (2, 3, 4)
C = np.array([[ [1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12] ],
               [ [13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]] ])

# Array 2: forma (4,)
D = np.array([1, 2, 3, 4])

# Moltiplicazione tra array N-dimensionali e array 1D (broadcasting)
result_nd = np.matmul(C, D)
```

# Condizioni con la Funzione where di NumPy

## Condizioni con `where`

- In NumPy, è possibile utilizzare la funzione `where` per applicare condizioni sugli elementi di un array.
- Dato un array di input, una condizione e due array, per ciascun elemento, l'output sarà campionato da `x` se la condizione è vera, altrimenti da `y`.

```
import numpy as np

# Array di input
arr = np.array([1, 2, 3, 4, 5, 6])

# Condizione: valori maggiori di 3
condition = arr > 3

# Array x e y
x = np.array([10, 20, 30, 40, 50, 60])
y = np.array([100, 200, 300, 400, 500, 600])

# Applicazione della funzione where
result = np.where(condition, x, y)

# Applicazione della funzione where
[100 200 300 40 50 60]
```

# Ottimizzazione con NumPy

## Considerazione

- NumPy è altamente ottimizzato per operazioni su array, quindi è consigliabile utilizzare il metodo "nativo" di NumPy quando possibile, invece di approcci di programmazione più tradizionali (es. i cicli).

## Esempio da Evitare

```
# Dimensioni dell'array
shape = (4, 640, 480, 3)

# Approccio tradizionale con cicli
start = time.time()
x = np.random.rand(*shape)
y = np.zeros_like(x)

for i in range(x.shape[0]):
    for j in range(x.shape[1]):
        for k in range(x.shape[2]):
            for l in range(x.shape[3]):
                if x[i, j, k, l] < 0.005:
                    y[i, j, k, l] = 0
                else:
                    y[i, j, k, l] = 255
duration_loop = time.time() - start
```

Tempo con Ciclo: 1.2431776523590088 s

# Ottimizzazione con NumPy

## Considerazione

- NumPy è altamente ottimizzato per operazioni su array, quindi è consigliabile utilizzare il metodo "nativo" di NumPy quando possibile, invece di approcci di programmazione più tradizionali (es. i cicli).

## Esempio Ottimale

```
start = time.time()
x = np.random.rand(*shape)
y_numpy = np.where(x < 0.005, 0, 255)
duration_numpy = time.time() - start
```

Tempo con Ciclo: 0.022099733352661133 s

- Utilizzare la funzionalità nativa di NumPy per le operazioni sugli array migliora le prestazioni e semplifica il codice.

# Memorizzazione e Caricamento di Array NumPy

## Salvataggio in File Binari

- NumPy consente di memorizzare e caricare array utilizzando file binari.
- Utilizzando `np.save`, è possibile serializzare gli array nel file system locale.
- Verrà creato un file con estensione `.npy` contenente l'array.

## Esempio Salvataggio

```
array = np.array([1, 2, 3, 4])
np.save('array_file.npy', array)
```

## Caricamento di File

- I file `.npy` possono essere letti utilizzando la funzione `np.load`.

## Esempio Caricamento

```
loaded_array = np.load('array_file.npy')
```

# **Libreria Matplotlib: Una Guida Rapida**

# Introduzione a Matplotlib

## Cos'è Matplotlib? ([Documentazione Online](#))

- Matplotlib è una libreria di plotting open source per Python.
- Permette di **creare grafici e diagrammi** di alta qualità in modo semplice e intuitivo.

## Caratteristiche Principali

- **Varietà di Grafici:** Supporta molti tipi di grafici, tra cui:
  - Grafici a linee
  - Istogrammi
  - Grafici a dispersione
  - Grafici 3D
- **Interattività:** Può funzionare in ambienti interattivi come Jupyter Notebook.
- **Integrazione con NumPy:** Utilizza array NumPy per gestire i dati, facilitando l'analisi scientifica.

## Installazione

- Matplotlib può essere installato facilmente con pip:

```
pip install matplotlib
```

# Visualizzazione con Matplotlib

## Grafici con Matplotlib

- Data una coppia di collezioni di valori, **m1** e **m2**, è possibile visualizzarli utilizzando Matplotlib.

```
import numpy as np
import matplotlib.pyplot as plt

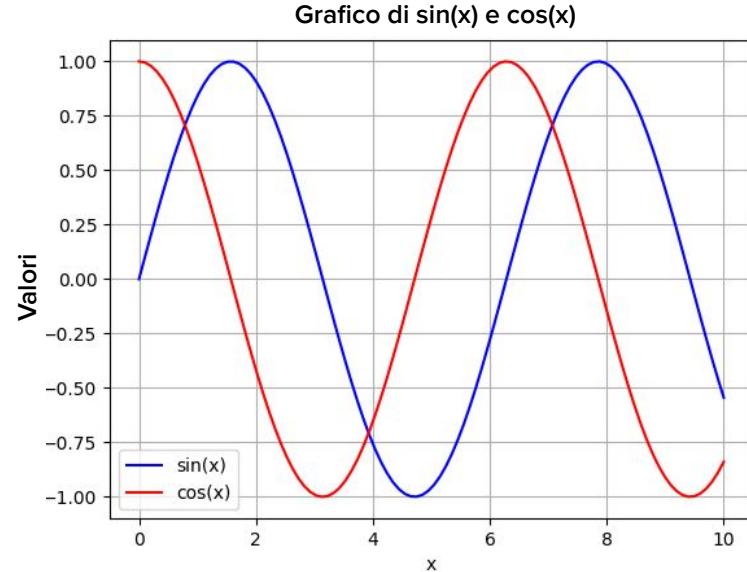
# Dati di esempio
x = np.linspace(0, 10, 100) # 100 punti tra 0 e 10
y1 = np.sin(x) # Andamento sinusoidale
y2 = np.cos(x) # Andamento coseno

# Creazione del grafico
plt.plot(x, y1, label='sin(x)', color='b')
plt.plot(x, y2, label='cos(x)', color='r')

# Aggiunta di titolo e etichette
plt.title('Grafico di sin(x) e cos(x)')
plt.xlabel('x')
plt.ylabel('Valori')
plt.grid()

# Aggiunta della legenda
plt.legend()

# Mostra il grafico
plt.show()
```



# Colormaps e Heatmaps

## Visualizzazione Dati Scientifici

- Le heatmap sono utili per visualizzare dati complessi, come la distribuzione della temperatura in una regione.
- Questa rappresentazione consente di osservare facilmente le variazioni di temperatura in diverse aree.

```
# 10x10 rappresenta la distribuzione della temperatura
# Temperature tra 0 e 30 gradi Celsius
temperature_data = np.random.rand(10, 10) * 30

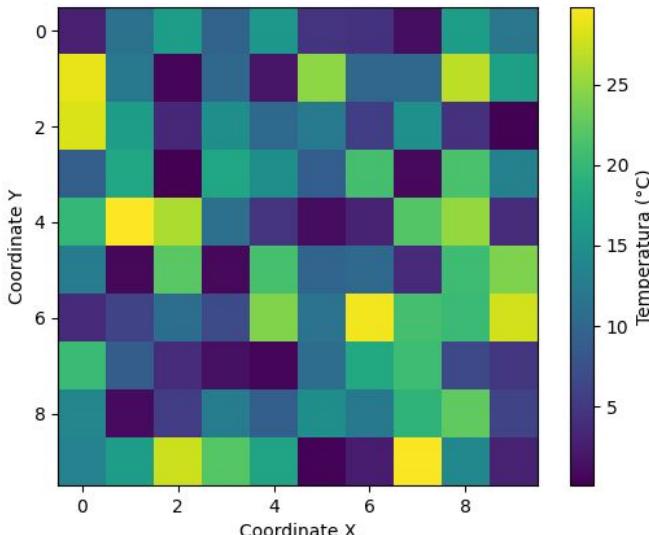
# Creazione della heatmap
plt.imshow(temperature_data, cmap='viridis',
           interpolation='nearest')

# Barra dei colori
plt.colorbar(label='Temperatura (°C)')

# Aggiunta di titolo e etichette
plt.title('Heatmap della Distribuzione della Temperatura')
plt.xlabel('Coordinate X')
plt.ylabel('Coordinate Y')

# Mostra la heatmap
plt.show()
```

Heatmap della Distribuzione della Temperatura



# Colormaps in Matplotlib

## Varietà di Colormaps

- Matplotlib offre una vasta gamma di **colormaps** per migliorare la **visualizzazione** dei dati.
- Le colormaps possono influenzare notevolmente l'**interpretazione** dei dati.
- **Documentazione Ufficiale:** <https://matplotlib.org/stable/users/explain/colors/colormaps.html>

## Tipi di Colormaps

- **Sequential:** Adatte per dati ordinati, come intensità o profondità.
  - Esempi: viridis, plasma, cividis
- **Diverging:** Utilizzate per rappresentare dati con un punto centrale significativo.
  - Esempi: coolwarm, bwr, RdBu
- **Qualitative:** Ideali per dati categorici, senza un ordine intrinseco.
  - Esempi: Set1, tab10, Paired

# Colormaps in Matplotlib

## Esempio di Codice per Visualizzare le Colormaps

```
import matplotlib.pyplot as plt
import numpy as np

# Creazione di un array di dati casuali
data = np.random.rand(10, 10)

# Visualizzazione di diverse colormaps
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Colormap Sequential
axes[0].imshow(data, cmap='viridis')
axes[0].set_title('Colormap: Viridis')

# Colormap Diverging
axes[1].imshow(data, cmap='coolwarm')
axes[1].set_title('Colormap: Coolwarm')

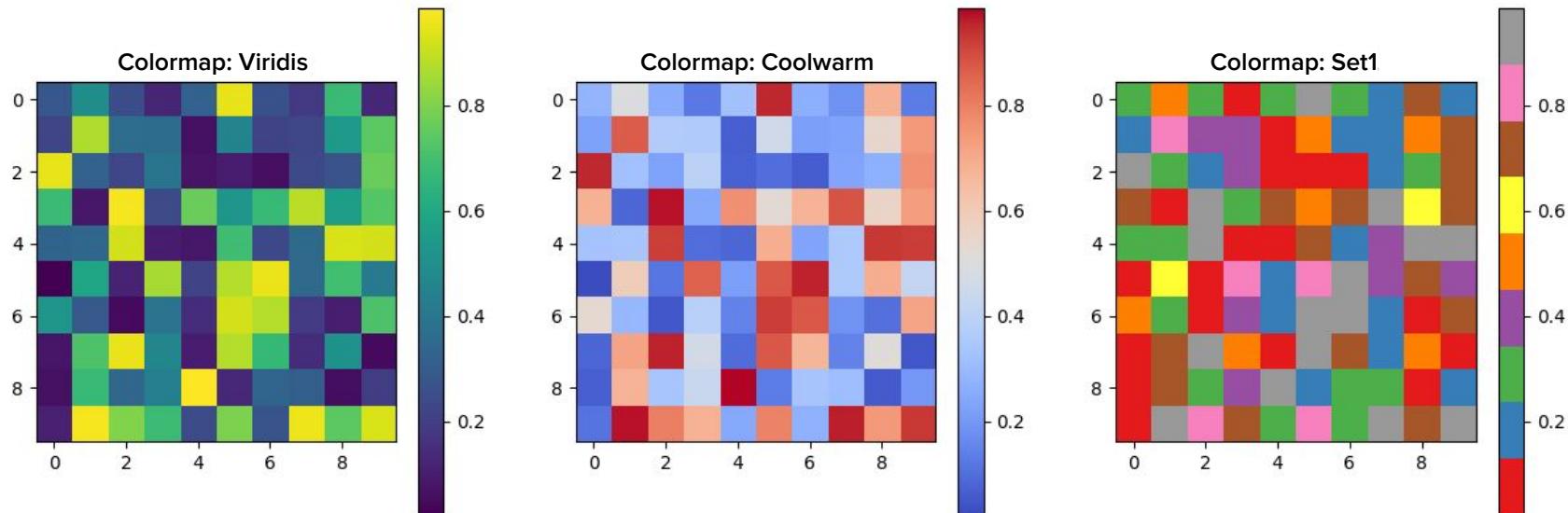
# Colormap Qualitative
axes[2].imshow(data, cmap='Set1')
axes[2].set_title('Colormap: Set1')

# Mostra le immagini
plt.colorbar(axes[0].imshow(data, cmap='viridis'), ax=axes[0])
plt.colorbar(axes[1].imshow(data, cmap='coolwarm'), ax=axes[1])
plt.colorbar(axes[2].imshow(data, cmap='Set1'), ax=axes[2])

plt.show()
```

# Colormaps in Matplotlib

## Risultato



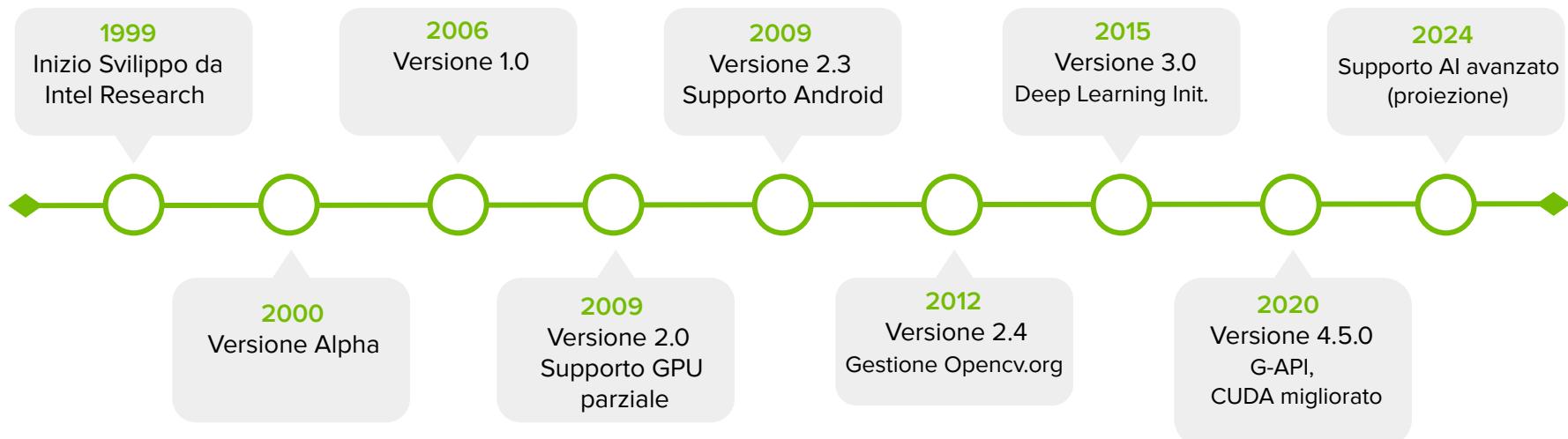
# **Libreria OpenCV: Una Guida Rapida**

# Introduzione a OpenCV



## Cosa è OpenCV? ([Documentazione Online](#))

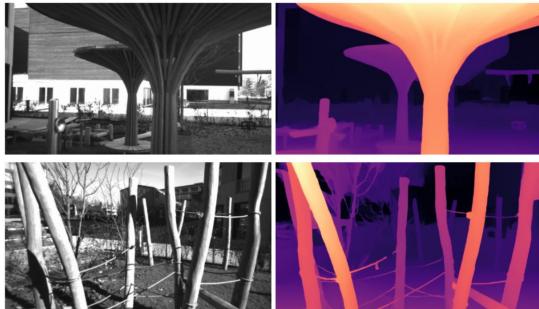
- OpenCV (Open Source Computer Vision Library) è una libreria software **open source** dedicata alla **computer vision** e al **machine learning**.
- Fornisce un'infrastruttura comune per applicazioni di computer vision.
- Supporta una vasta gamma di **linguaggi di programmazione**, tra cui Python, C++, Java e MATLAB.
- Può essere installata su Windows, Linux, Android e Mac OS



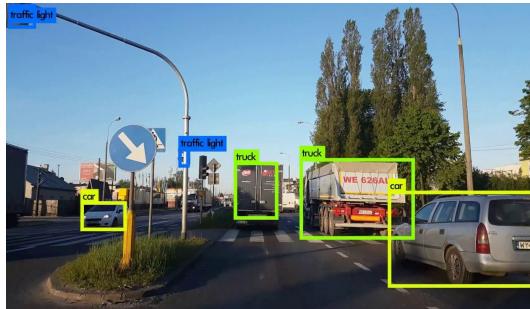
# Importanza nella Computer Vision



- Offre oltre 2500 algoritmi ottimizzati per compiti di computer vision e machine learning.



Stereo Matching



Object Detection



Image Segmentation



Feature Detection



Optical Flow

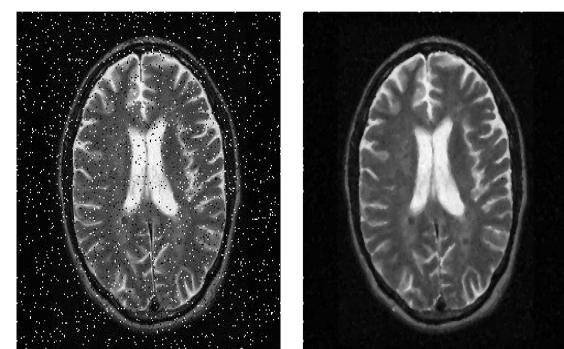


Image Filtering

# Moduli Principali di OpenCV



- OpenCV è organizzato in vari moduli che forniscono strumenti e funzioni specializzate per diverse applicazioni di visione artificiale.

## Core

- Operazioni di base di elaborazione immagini e array
- Strutture dati fondamentali (Mat, Vector, Scalar)
- Funzioni matematiche di base

[Documentazione](#)

## Imgproc

- Elaborazione di immagini
- Filtraggio, trasformazioni geometriche
- Istogrammi, edge detection, morfologia

[Documentazione](#)

## Video

- Analisi video
- Background subtraction, object tracking
- Optical flow

[Documentazione](#)

## Calib3d

- Calibrazione della camera
- Ricostruzione 3D
- Geometria stereo
- Rimozione della Distorsione

[Documentazione](#)

## Features2d

- Rilevamento di feature
- Descrittori di feature
- Feature matching

[Documentazione](#)

## Highgui

- Interfaccia grafica per la visualizzazione
- Acquisizione di immagini/video
- Creazione di finestre e gestione degli eventi

[Documentazione](#)

# Moduli Principali di OpenCV



- OpenCV è organizzato in vari moduli che forniscono strumenti e funzioni specializzate per diverse applicazioni di visione artificiale.

## Objdetect

- Rilevamento di oggetti
- Cascade classifiers
- HOG detectors
- Tracking di Oggetti

[Documentazione](#)

## Stitching

- Image stitching
- Creazione di panorami
- Riconoscimento di Scene
- Correzione dell'Illuminazione

[Documentazione](#)

## Contrib

- Moduli sperimentali o in fase di sviluppo
- Algoritmi e funzionalità avanzate

[Documentazione](#)

## DNN

- Deep learning
- Caricamento e inferenza su modelli pre-addestrati
- Supporto per framework come Caffe, TensorFlow, PyTorch

[Documentazione](#)

## ML

- Algoritmi di apprendimento automatico
- SVM, decision trees, k-means, random forests

[Documentazione](#)

# Applicazioni

- OpenCV, con la sua vasta gamma di algoritmi per la visione artificiale, alimenta una miriade di applicazioni



Sistemi di sorveglianza



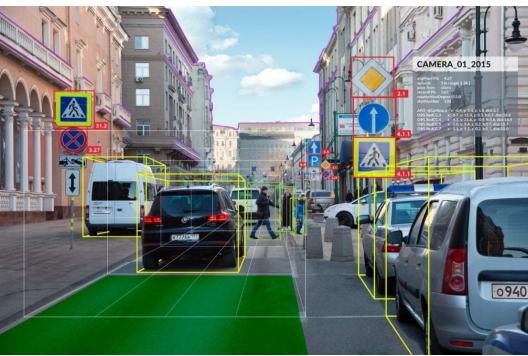
Robotica



Realtà Virtuale



Analisi Medica



Guida Autonoma



Realtà Aumentata

# Installazione di OpenCV con Python



## 1. Creare un Ambiente Virtuale

- Perché un ambiente virtuale?
  - **Isolamento:** Evita conflitti tra librerie di progetti diversi.
  - **Pulizia:** Mantiene il sistema operativo pulito da librerie non necessarie.
  - **Riproducibilità:** Garantisce la compatibilità tra ambienti di sviluppo e produzione.
- Aprire il terminale e installare virtualenv (se non è già installato)

```
pip install virtualenv
```

- Crea un nuovo ambiente virtuale:

```
virtualenv myenv
```

- Attiva l'ambiente virtuale:

- Windows

```
myenv\Scripts\activate
```

- Linux

```
source myenv/bin/activate
```

# Installazione di OpenCV con Python



## 1. Installa OpenCV nell'Ambiente Virtuale:

- Una volta attivato l'ambiente, installa **OpenCV** e **numpy** con il gestore di pacchetti pip:

```
pip install opencv-python numpy
```

- Per includere moduli aggiuntivi (contrib),

```
pip install opencv-contrib-python
```

## 2. Verifica dell'installazione

- Apri un interprete python nel terminale:

```
python
```

- Importa OpenCV per verificare:

```
import cv2  
print(cv2.__version__)
```

- Se il numero di versione (e.g. 4.8.0) viene visualizzato correttamente, l'installazione è riuscita

# OpenCV: Lettura e Scrittura delle Immagini



## Esempio di Codice

- OpenCV offre funzioni per **leggere** e **scrivere** immagini.
- È possibile aprire un'immagine utilizzando la funzione **imread**.
- Inoltre, gestisce vari **formati** di immagine (PNG, JPEG, ecc.) e **tipi di dati** (8 bit, 16 bit, ecc.).

```
import cv2

# Caricare un'immagine
img = cv2.imread('percorso/della/tua/immagine.jpg')

# Visualizzare l'immagine
cv2.imshow('Immagine', img)

# Attendere un tasto e chiudere la finestra
cv2.waitKey(0)
cv2.destroyAllWindows()

# Scrivere un'immagine su disco
cv2.imwrite('percorso/dove/salvare/immagine.jpg', img) # Salva l'immagine
```

# OpenCV: Gestione delle Immagini



## Gestione Immagini

- OpenCV restituisce un array NumPy per le immagini (ogni valore rappresenta un pixel)
- Il formato predefinito è **BGR**, quindi è necessario convertire in **RGB** utilizzando cvtColor.
- Può anche trasformare immagini a colori in scala di grigi con cv2.BGR2GRAY o cv2.RGB2GRAY.

```
import cv2

# Caricare e convertire un'immagine
img = cv2.imread('percorso/della/tua/immagine.jpg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Visualizzare le immagini
cv2.imshow('RGB', img_rgb)
cv2.imshow('Grayscale', img_gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

# OpenCV: Gestione delle Immagini



## Gestione Immagini

- OpenCV restituisce un array NumPy per le immagini (ogni valore rappresenta un pixel)
- Il formato predefinito è **BGR**, quindi è necessario convertire in **RGB** utilizzando `cvtColor`.
- Può anche trasformare immagini a colori in scala di grigi con `cv2.BGR2GRAY` o `cv2.RGB2GRAY`.



RGB



BGR



GRAY

# Scrittura di un'Immagine con OpenCV



## Scrittura

- Si può scrivere un'immagine nel file system utilizzando la funzione **imwrite**.
- OpenCV si aspetta un'immagine in formato BGR; quindi, se l'immagine (img) è in formato RGB, è necessario convertirla in BGR prima di salvare, utilizzando **cv2.cvtColor**.

## Esempio di Codice

```
import cv2
import numpy as np

# Crea un'immagine fittizia RGB (3 canali)
img_rgb = np.random.randint(0, 255, (100, 100, 3), dtype=np.uint8)

# Converti da RGB a BGR
img_bgr = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)

# Salva l'immagine in formato JPG
cv2.imwrite('percorso/della/tua/immagine_bgr.jpg', img_bgr)
```

# Ridimensionamento delle Immagini



## Resize di una Immagine

- OpenCV consente di **ridimensionare** le immagini utilizzando la funzione **resize**.
- Questa funzione richiede l'immagine da ridimensionare e la **nuova shape** desiderata.

## Sintassi

- **image**: l'immagine da ridimensionare.
- **dsize**: la nuova dimensione dell'immagine come tupla (**larghezza, altezza**).

## Esempio di Codice

```
import cv2

# Caricare un'immagine
img = cv2.imread('percorso/della/tua/immagine.jpg')

# Ottenere le dimensioni originali
height, width = img.shape[:2]

# Ridimensionare l'immagine a 1/4 della risoluzione originale (oppure arbitrarie)
img_resized = cv2.resize(img, (width // 4, height // 4))
```

# Ridimensionamento delle Immagini



## Resize di una Immagine

- OpenCV consente di **ridimensionare** le immagini utilizzando la funzione **resize**.
- Questa funzione richiede l'immagine da ridimensionare e la **nuova shape** desiderata.

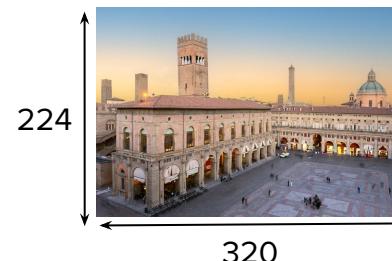
## Sintassi

- **image**: l'immagine da ridimensionare.
- **dsize**: la nuova dimensione dell'immagine come tupla (**larghezza, altezza**).

Immagine Originale



Immagine  
Ridimensionata



# Trasformazione Lineare delle Immagini con OpenCV

## Modifica dell'Intensità

- La **trasformazione lineare** è una tecnica per manipolare i valori dei pixel di un'immagine.
- Utilizza la formula:  $\text{output} = \alpha * \text{input} + \beta$

## Elementi Chiave

- **Parametri**
  - **$\alpha$  (alpha)**: controlla il contrasto
  - **$\beta$  (beta)**: controlla la luminosità
- **Effetti**
  - $\alpha > 1$ : Aumenta il contrasto
  - $0 < \alpha < 1$ : Diminuisce il contrasto
  - $\beta > 0$ : Aumenta la luminosità
  - $\beta < 0$ : Diminuisce la luminosità

## Esempio di Codice

```
def adjust_brightness_contrast(image, alpha=1.0, beta=0):
    return cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

# Esempio d'uso
image = cv2.imread('percorso/della/tua/immagine.jpg')
adjusted = adjust_brightness_contrast(image, alpha=1.5, beta=30)
```

# Trasformazione Lineare delle Immagini con OpenCV



Immagine Originale



Brighter,  $\alpha=1.0$ ,  $\beta=50$



Darker,  $\alpha=1.0$ ,  $\beta=-50$



Higher Contrast,  $\alpha=1.5$ ,  $\beta=0$



Lower Contrast,  $\alpha=0.5$ ,  $\beta=0$

# Conversione da RGB a Grayscale con OpenCV

## Conversione RGB to Gray

- La conversione di un'immagine da RGB a grayscale trasforma un'immagine a colori in un'immagine in scala di grigi, dove ogni pixel è rappresentato da un singolo valore di intensità luminosa.

## Formula di Conversione

$$\text{Gray} = 0.299\text{R} + 0.587\text{G} + 0.114\text{B} \text{ (per pixel)}$$

## Esempio di Codice

### Metodo 1

```
# Carica l'immagine a colori
img = cv2.imread('percorso/della/tua/immagine.jpg')

# Converti l'immagine in scala di grigi
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
```

### Metodo 2

```
# Carica direttamente l'immagine in scala di grigi
img_gray_direct = cv2.imread('percorso/della/tua/immagine.jpg', 0)
```

# Conversione da RGB a Grayscale con OpenCV

## Conversione RGB to Gray

- La conversione di un'immagine da RGB a grayscale trasforma un'immagine a colori in un'immagine in scala di grigi, dove ogni pixel è rappresentato da un singolo valore di intensità luminosa.

## Formula di Conversione

$$\text{Gray} = 0.299\text{R} + 0.587\text{G} + 0.114\text{B} \text{ (per pixel)}$$



Immagine Originale



Immagine Grayscale

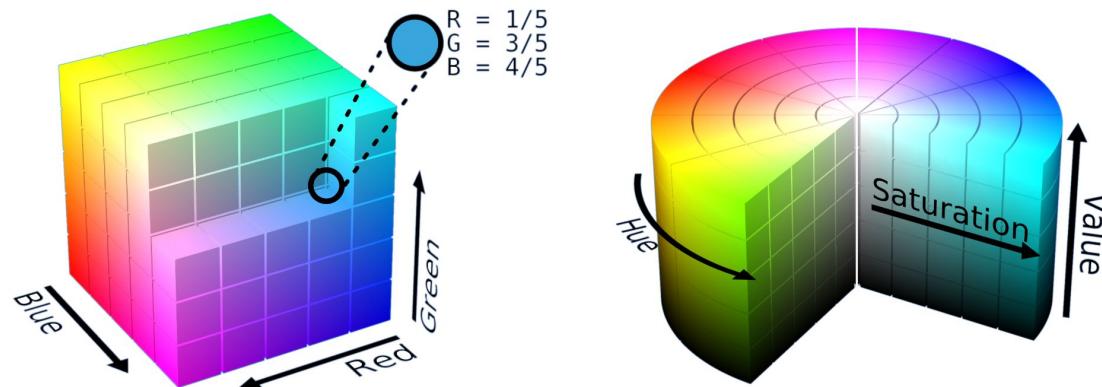
# Trasformazione di Colore

## Introduzione alla Trasformazione di Colore

- La **trasformazione di colore** è il processo di conversione di un'immagine da uno spazio colore a un altro.
- **Scopo:** migliorare l'analisi, l'elaborazione e la percezione delle immagini digitali.

## Spazi Colore Comuni:

- **RGB (Red, Green, Blue):** Usato in display digitali. Additivo, basato sui colori primari della luce
- **HSV (Hue, Saturation, Value):** Più intuitivo per la percezione umana. Utile per la segmentazione del colore.
- **LAB:** Progettato per essere percettivamente uniforme. Separa la luminosità (L) dal colore (a, b)
- **Scala di grigi:** Rappresenta solo l'intensità luminosa. Utile per ridurre la complessità e l'elaborazione



# Trasformazione di Colore

## Introduzione alla Trasformazione di Colore

- La **trasformazione di colore** è il processo di conversione di un'immagine da uno spazio colore a un altro.
- **Scopo:** migliorare l'analisi, l'elaborazione e la percezione delle immagini digitali.

## Spazi Colore Comuni:

- **RGB (Red, Green, Blue):** Usato in display digitali. Additivo, basato sui colori primari della luce
- **HSV (Hue, Saturation, Value):** Più intuitivo per la percezione umana. Utile per la segmentazione del colore.
- **LAB:** Progettato per essere percettivamente uniforme. Separa la luminosità (L) dal colore (a, b)
- **Scala di grigi:** Rappresenta solo l'intensità luminosa. Utile per ridurre la complessità e l'elaborazione

## Esempio di Codice

```
# Carica l'immagine
image = cv2.imread('percorso/della/tua/immagine.jpg')

# Converti in RGB
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Converti in scala di grigi
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Converti in HSV
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

# Trasformazione di Colore

## Introduzione alla Trasformazione di Colore

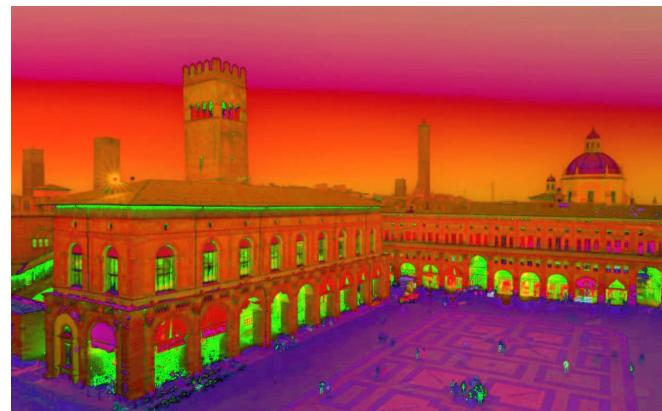
- La **trasformazione di colore** è il processo di conversione di un'immagine da uno spazio colore a un altro.
- **Scopo:** migliorare l'analisi, l'elaborazione e la percezione delle immagini digitali.

## Spazi Colore Comuni:

- **RGB (Red, Green, Blue):** Usato in display digitali. Additivo, basato sui colori primari della luce
- **HSV (Hue, Saturation, Value):** Più intuitivo per la percezione umana. Utile per la segmentazione del colore.
- **LAB:** Progettato per essere percettivamente uniforme. Separa la luminosità (L) dal colore (a, b)
- **Scala di grigi:** Rappresenta solo l'intensità luminosa. Utile per ridurre la complessità e l'elaborazione



RGB



HSV

# Istogramma di un'Immagine

## Introduzione all'Iistogramma

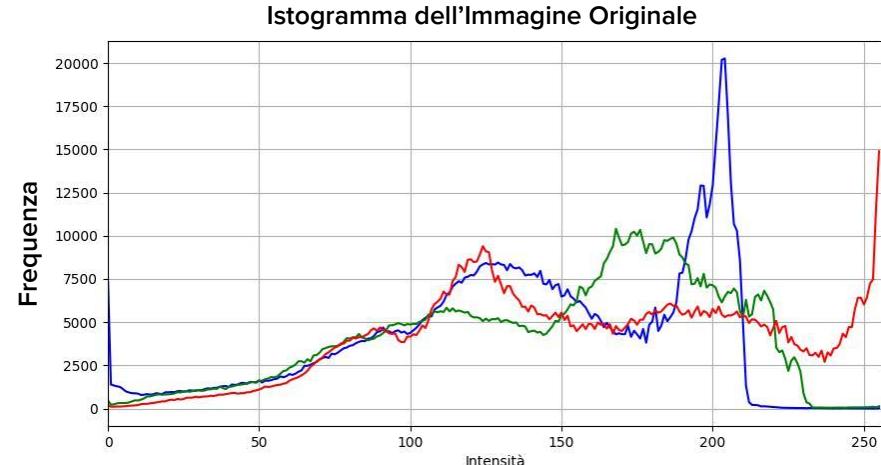
- Un istogramma è una rappresentazione grafica della **distribuzione dei valori di intensità** di un'immagine.
- Mostra la **frequenza** di ogni livello di intensità, fornendo informazioni utili sul contrasto e sul contenuto dell'immagine.

## Caratteristiche

- **Asse X:** Rappresenta i livelli di intensità (da 0 a 255 per immagini a 8 bit).
- **Asse Y:** Indica la frequenza di ciascun livello di intensità, ossia il numero di pixel corrispondenti.



RGB



# Equalizzazione dell'Istogramma con OpenCV

## Introduzione all'Equalizzazione dell'Istogramma

- L'**equalizzazione dell'istogramma** è una tecnica di elaborazione delle immagini che migliora il contrasto **distribuendo** meglio i livelli di intensità dell'immagine. È spesso utilizzata per:
  - **Migliorare i dettagli** nelle immagini a basso contrasto.
  - Rendere le variazioni di intensità più **uniformi** su tutta l'immagine.

## Processo

- **Calcolo dell'Istogramma:** Rappresenta la frequenza di ogni livello di intensità (0-255).
- **Funzione di Distribuzione Cumulativa (CDF):** Si calcola la CDF dell'istogramma. La CDF accumula le frequenze, fornendo un valore per ogni livello di intensità che rappresenta la somma dei pixel fino a quel livello.
- **Normalizzazione:** La CDF viene normalizzata dividendo ciascun valore per il numero totale di pixel. Questo produce una funzione che varia da 0 a 1.
- **Mappatura dei Pixel:** I pixel dell'immagine originale vengono mappati ai nuovi livelli di intensità usando la CDF normalizzata, moltiplicata per il valore massimo (255).

## Note

- L'equalizzazione può essere applicata a immagini in **scala di grigi**, elaborando il singolo canale di intensità, o a **immagini a colori**, utilizzando il canale di luminanza in spazi di colore come YUV o HSV.

# Equalizzazione dell'Istogramma con OpenCV

## Introduzione all'Equalizzazione dell'Istogramma

- L'equalizzazione dell'istogramma è una tecnica di elaborazione delle immagini che migliora il contrasto distribuendo meglio i livelli di intensità dell'immagine. È spesso utilizzata per:
  - Migliorare i dettagli nelle immagini a basso contrasto.
  - Rendere le variazioni di intensità più uniformi su tutta l'immagine.



Equalizzazione



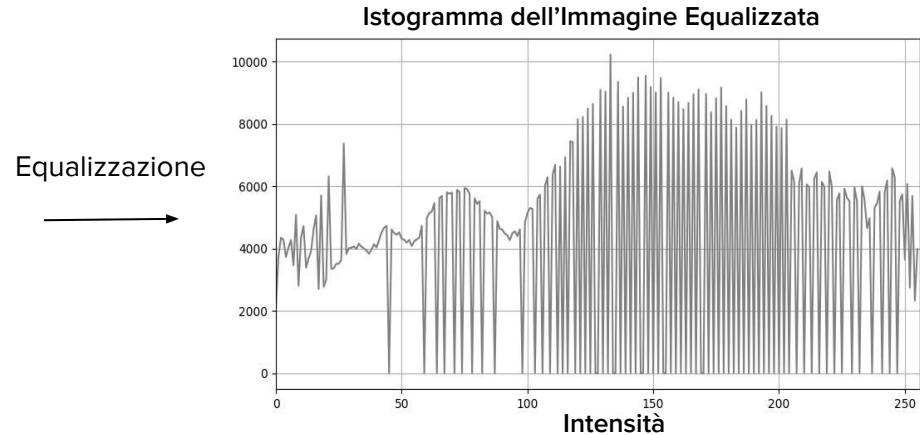
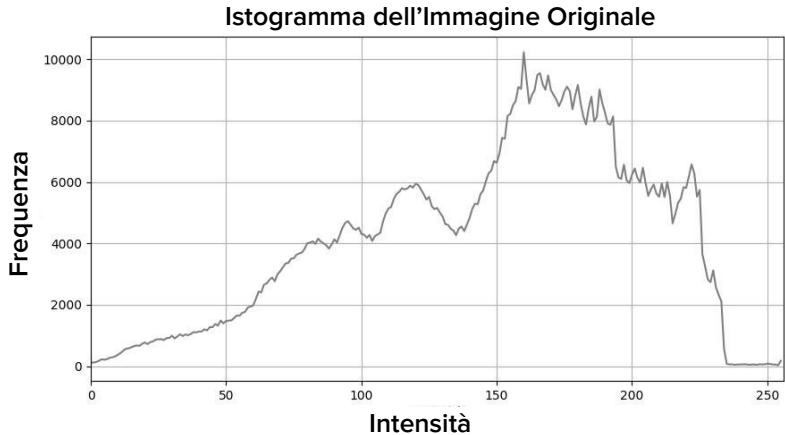
Immagine Originale Grayscale

Immagine Equalizzata

# Equalizzazione dell'Istogramma con OpenCV

## Introduzione all'Equalizzazione dell'Istogramma

- L'equalizzazione dell'istogramma è una tecnica di elaborazione delle immagini che migliora il contrasto distribuendo meglio i livelli di intensità dell'immagine. È spesso utilizzata per:
  - Migliorare i dettagli nelle immagini a basso contrasto.
  - Rendere le variazioni di intensità più uniformi su tutta l'immagine.



# Flipping di un'Immagine

## Introduzione all'Image Flipping

- L'image flipping è una tecnica di elaborazione delle immagini che inverte l'ordine dei pixel lungo un asse specifico per ciascun canale di colore, creando un **effetto specchio**. Il flipping può essere:
  - Orizzontale:** Invertendo l'ordine dei pixel da sinistra a destra.
  - Verticale:** Invertendo l'ordine dei pixel dall'alto verso il basso.



Input Image



Flip Orizzontale



Flip Verticale

# Flipping di un'Immagine

## Introduzione all'Image Flipping

- L'image flipping è una tecnica di elaborazione delle immagini che inverte l'ordine dei pixel lungo un asse specifico per ciascun canale di colore, creando un **effetto specchio**. Il flipping può essere:
  - **Orizzontale:** Invertendo l'ordine dei pixel da sinistra a destra.
  - **Verticale:** Invertendo l'ordine dei pixel dall'alto verso il basso.

## Esempio di Codice

```
import cv2

# Carica l'immagine
image = cv2.imread('percorso/della/tua/immagine.jpg')

# Flipping orizzontale
horizontal_flip = cv2.flip(image, 1)

# Flipping verticale
vertical_flip = cv2.flip(image, 0)
```

# Image Blur con OpenCV

## Introduzione all'Image Blurring

L'image blurring è una tecnica di elaborazione delle immagini che riduce i dettagli e le variazioni di intensità, creando un **effetto di sfocatura**. Viene utilizzata per:

- **Riduzione del rumore:** Attenuando le fluttuazioni casuali dei pixel.
- **Enfasi degli oggetti:** Sfumando i dettagli irrilevanti e mettendo in risalto gli elementi principali.
- **Preprocessing per la computer vision:** Semplificando l'immagine per facilitarne l'analisi da parte degli algoritmi.



Input Image



Blurred Image (kernel\_size=25)

# Image Blur con OpenCV

## Concetto di Base

Il blurring viene realizzato attraverso il calcolo della **media dei valori** di intensità dei pixel circostanti. L'operazione può essere riassunta come segue:

- **Patch di dimensioni N×N:** Una finestra di dimensioni fisse scorre su ciascun pixel dell'immagine.
- **Pixel centrale:** Ogni pixel di output è la media dei pixel nella patch che lo circondano.
- **Esempio con patch 3×3:** Include il pixel centrale più gli 8 pixel che lo circondano, formando una matrice di 3 righe e 3 colonne.

## Esempio di Codice

```
# Carica l'immagine
image = cv2.imread('percorso/della/tua/immagine.jpg')

# Definisce la dimensione del kernel per il blurring (es. 3x3)
kernel_size = (3, 3)

# Applica il blurring utilizzando il filtro di media con il kernel 3x3
blurred_image = cv2.blur(image, kernel_size)
```

# Introduzione alla Convoluzione 1D e 2D

## Che cos'è la Convoluzione?

- Operazione matematica lineare tra **due funzioni**, segnale e kernel (fourvante - spesso indicato come **filtro**).
- Misura la **sovraposizione** del filtro con il segnale mentre scorre su di esso.
- Produce una nuova funzione (segnale di output) che rappresenta le **caratteristiche estratte** dal segnale di input.

## Convoluzione 1D

- Applicata a **dati unidimensionali** (segnali audio, serie temporali, sequenze di testo).
- Il filtro è un vettore che **scorre** sul segnale.
- L'output ad ogni punto è la **somma dei prodotti elemento per elemento (prodotto scalare)** tra il filtro e la porzione di segnale sottostante.
- **Esempio:** Applicazione di un filtro di media mobile su un segnale audio per ridurre il rumore.

## Convoluzione 2D

- Applicata a **dati bidimensionali** (es. immagini).
- Il filtro è una matrice che **scorre** sull'immagine.
- L'output ad ogni pixel è la **somma dei prodotti elemento per elemento (prodotto scalare)** tra il filtro e la regione dell'immagine sottostante.
- **Esempio:** (Image Blur caso particolare di convoluzione 2D. Perchè?)
  - Applicazione di un filtro di **rilevamento dei bordi** a un'immagine per estrarre i contorni degli oggetti.
  - Fondamentale nelle **reti neurali convoluzionali (CNN)** per l'elaborazione di immagini.

# Convoluzione 2D: Rilevazione dei Contorni con Sobel

## Processo di Convoluzione 2D

- Il kernel (o filtro) di **Sobel** scorre sull'immagine pixel per pixel
- Per ogni posizione
  - Si **moltiplica** il kernel per la regione corrispondente dell'immagine
  - Si **somma** i risultati per ottenere il valore del pixel di output
- Evidenzia le **transizioni verticali** di intensità nell'immagine

Kernel di Sobel  
Verticale

-1	0	1
-2	0	2
-1	0	1

## Esempio di Codice

```
import cv2
import numpy as np

# Carica l'immagine in scala di grigi
image = cv2.imread('percorso/della/tua/immagine.jpg', cv2.IMREAD_GRAYSCALE)

# Applica il filtro Sobel verticale (dx=1, dy=0) per rilevare i contorni verticali
sobel_vertical = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)

# Converte i risultati in valori assoluti e in formato 8 bit per la visualizzazione
sobel_vertical_abs = cv2.convertScaleAbs(sobel_vertical)
```

# Convoluzione 2D: Rilevazione dei Contorni con Sobel

## Processo di Convoluzione 2D

- Il kernel (o filtro) di **Sobel** scorre sull'immagine pixel per pixel
- Per ogni posizione
  - Si **moltiplica** il kernel per la regione corrispondente dell'immagine
  - Si **somma** i risultati per ottenere il valore del pixel di output
- Evidenzia le **transizioni orizzontali** di intensità nell'immagine

Kernel di Sobel  
Orizzontale

-1	-2	-1
0	0	0
-1	-2	-1

## Esempio di Codice

```
import cv2
import numpy as np

# Carica l'immagine in scala di grigi
image = cv2.imread('percorso/della/tua/immagine.jpg', cv2.IMREAD_GRAYSCALE)

# Applica il filtro Sobel verticale (dx=1, dy=0) per rilevare i contorni verticali
sobel_horizontal = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)

# Converte i risultati in valori assoluti e in formato 8 bit per la visualizzazione
sobel_horizontal_abs = cv2.convertScaleAbs(sobel_horizontal)
```

# Convoluzione 2D: Rilevazione dei Contorni con Sobel

## Processo di Convoluzione 2D

- Il kernel (o filtro) di **Sobel** scorre sull'immagine pixel per pixel
- Per ogni posizione
  - Si **moltiplica** il kernel per la regione corrispondente dell'immagine
  - Si **somma** i risultati per ottenere il valore del pixel di output
- Evidenzia le **transizioni verticali** di intensità nell'immagine

Kernel di Sobel  
Verticale

-1	0	1
-2	0	2
-1	0	1



Input Image



Contorni Verticali

# Convoluzione 2D: Rilevazione dei Contorni con Sobel

## Processo di Convoluzione 2D

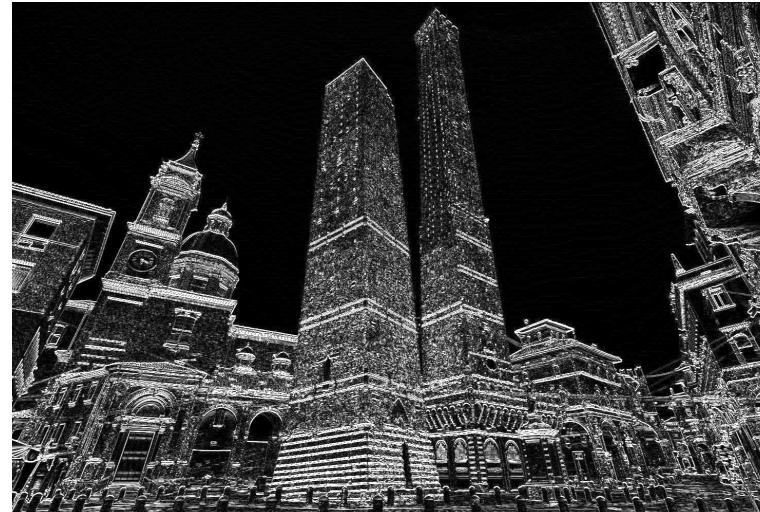
- Il kernel (o filtro) di **Sobel** scorre sull'immagine pixel per pixel
- Per ogni posizione
  - Si **moltiplica** il kernel per la regione corrispondente dell'immagine
  - Si **somma** i risultati per ottenere il valore del pixel di output
- Evidenzia le **transizioni verticali** di intensità nell'immagine

Kernel di Sobel  
Orizzontale

-1	-2	-1
0	0	0
-1	-2	-1



Input Image



Contorni Orizzontali

# Riferimenti Bibliografici e Link Utili



## Python - Riferimenti & Link Utili

- Marco Buttu (2014): **Programmare con Python. Guida completa.** Edizioni Lswr; 1° edizione
- **Guida Python:** <https://www.html.it/guide/guida-python/>
- **Tutorial:** <https://docs.python.org/it/3/tutorial/index.html>

## OpenCV - Link Utili

- **Home:** <https://opencv.org/>
- **Documentazione:** <https://docs.opencv.org/>
- **Q&A Forum:** <http://answers.opencv.org/>
- **GitHub:** <https://github.com/opencv/>