# Sviluppo Didattico in Cloud con Python e Gradio

Manuale Operativo e Soluzioni Svolte

Febbraio 2026

## Indice

# 1 Introduzione all'Ambiente Cloud

Per uno sviluppatore proveniente da ambienti enterprise (Java) o web frontend (JavaScript), l'approccio di Python in cloud su piattaforme come Hugging Face richiede un cambio di paradigma minimo ma essenziale. Non vi è un processo di build pesante né un server application (come Tomcat o Node.js) da configurare manualmente.

## 1.1 Le Dipendenze: Il file `requirements.txt`

L'equivalente del `pom.xml` (Maven) o del `package.json` (npm) in Python è il file `requirements.txt`. Il server legge questo file testuale prima di lanciare l'applicazione e installa i pacchetti necessari tramite il gestore `pip`.

Per gli esempi di questo manuale, il file `requirements.txt` deve contenere esattamente:

```
gradio
matplotlib
numpy
```

# 2 Soluzioni Svolte: Gestione Dati e Interfacce

L'architettura di base usa la libreria `Gradio` come "View" e Python puro come "Controller/Model". Di seguito i moduli applicativi.

## 2.1 Caso Studio 1: Il Filesystem

In cloud, è possibile leggere e scrivere file testuali (CSV, log, txt) esattamente come in locale. L'esempio seguente simula l'inserimento di un record e il conteggio delle righe totali.

```
import gradio as gr
import os

def gestisci_file(testo):
    # Scrittura in append mode ("a")
    with open("database_didattico.txt", "a") as f:
        f.write(testo + "\n")

    # Lettura delle righe
    with open("database_didattico.txt", "r") as f:
        righe = f.readlines()

    return f"Record salvato. Totale voci nel file: {len(righe)}"

gr.Interface(
    fn=gestisci_file,
    inputs=gr.Textbox(label="Nuovo Record"),
    outputs="text",
    title="Simulatore di Filesystem"
).launch()
```

Listing 1: App per la gestione del filesystem

## 2.2 Caso Studio 2: Grafica Matematiche (Ex-Turtle)

L'approccio grafico nativo (Turtle/Tkinter) non è utilizzabile su server senza display X11. La strategia didattica adottata prevede l'uso di `matplotlib` per generare grafici statici o interattivi basati su funzioni matematiche, restituendo un oggetto *Plot*.

```python
import gradio as gr
import matplotlib.pyplot as plt
import numpy as np

def plot_funzione(frequenza):
    x = np.linspace(0, 10, 500)
    y = np.sin(frequenza * x)

    fig, ax = plt.subplots()
    ax.plot(x, y, color="blue")
    ax.set_title(f"Onda Sinusoidale con Frequenza = {frequenza}")
    ax.grid(True)
    return fig

gr.Interface(
    fn=plot_funzione,
    inputs=gr.Slider(1, 10, label="Modula la Frequenza"),
    outputs=gr.Plot(label="Oscilloscopio Matematico"),
    title="Generatore di Onde"
).launch()
```

Listing 2: Tracciamento di un'onda sinusoidale

### 2.3 Caso Studio 3: Animazioni di Algoritmi (Bubble Sort)

Non potendo usare animazioni video dirette, simuliamo il trascorrere del tempo con uno *slider di stato*. L'algoritmo calcola tutti i passaggi a monte e l'interfaccia permette di scorrere gli "step" temporali.

```python
import gradio as gr
import matplotlib.pyplot as plt

def esegui_bubble_sort(lista_str):
    arr = [int(x.strip()) for x in lista_str.split(",")]
    steps = [list(arr)] # Stato iniziale

    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j] # Swap
                steps.append(list(arr)) # Salva snapshot
    return steps

def visualizza_step(lista_input, step_index):
    tutti_gli_step = esegui_bubble_sort(lista_input)
    idx = min(step_index, len(tutti_gli_step)-1)
    current_arr = tutti_gli_step[idx]

    fig, ax = plt.subplots()
    ax.bar(range(len(current_arr)), current_arr, color="skyblue")
    ax.set_title(f"Snapshot dell'algoritmo al passaggio {idx}")
    return fig

with gr.Blocks() as demo:
    gr.Markdown("### Animatore di Ordinamento")
    input_txt = gr.Textbox(value="10, 2, 8, 4, 6", label="Array (CSV)")
    slider = gr.Slider(minimum=0, maximum=20, step=1, label="Scorri gli
   Step")
```

```
    plot = gr.Plot()

    input_txt.change(visualizza_step, [input_txt, slider], plot)
    slider.change(visualizza_step, [input_txt, slider], plot)

if __name__ == "__main__":
    demo.launch()
```

Listing 3: Animatore interattivo di Bubble Sort

# 3 Sfide Didattiche: Esercizi Aperti di Logica

Gli esercizi seguenti richiedono la progettazione dell'algoritmo interno in Python e dell'interfaccia Gradio, senza fornire soluzioni preconfezionate. L'obiettivo è tradurre un requisito logico in codice.

## 3.1 Esercizio A: Il Database Persistente a Dizionario

**Obiettivo:** Sviluppare un'app che si comporti come una mini-tabella SQL basata su file testuali.

- **Input:** Due campi di testo (Es: "Nome Studente" e "Voto"). Un bottone "Salva" e un bottone "Cerca".
- **Logica:** Quando si clicca Salva, i dati vengono scritti in un file `voti.csv`. Se si usa la funzione Cerca inserendo un nome, l'app deve scorrere il file, trovare la riga corrispondente e restituire il voto.
- **Difficoltà:** Gestire l'assenza del file (se non è ancora stato creato) e gestire i record non trovati o duplicati.

## 3.2 Esercizio B: Sintesi di Forme d'Onda Complesse

**Obiettivo:** Estendere il Caso Studio 2 per generare graficamente un segnale composto (utile per introdurre concetti fisici e acustici).

- **Input:** Tre slider: "Frequenza Base", "Armonica", "Ampiezza Armonica".
- **Logica:** Generare un array matematico che sia la somma di due onde:

$$y = \sin(f_{base} \cdot x) + A_{arm} \cdot \sin(f_{arm} \cdot x)$$

- **Output:** Il grafico Matplotlib risultante che si aggiorna in tempo reale.

## 3.3 Esercizio C: Animatore Ricorsivo (Quick Sort)

**Obiettivo:** Adattare la logica dello slider temporale (Caso Studio 3) a un algoritmo di ordinamento ricorsivo.

- **Difficoltà:** Nel Bubble Sort è facile salvare uno snapshot dentro a un doppio ciclo `for`. Nel Quick Sort (o Merge Sort), la natura ricorsiva rende più complessa l'estrazione degli "stati intermedi" della lista per passarli al visualizzatore Matplotlib.
- **Sfida:** Implementare una variabile globale o una classe che registri una copia (`list(arr)`) della lista ad ogni spostamento del *pivot*.