Ambiente di Sviluppo Python: Fondamenti per Principianti

Benvenuti al nostro corso introduttivo sugli ambienti di sviluppo Python! Nei prossimi giorni, esploreremo come configurare correttamente un ambiente di sviluppo professionale, gestire dipendenze e organizzare progetti in modo efficace.

Questa presentazione è pensata per chi ha iniziato a programmare in Python da poco e desidera apprendere le migliori pratiche per organizzare il proprio lavoro. Partiremo dalla struttura base di un progetto e arriveremo alla condivisione di ambienti completi con altri sviluppatori.

Iniziamo questo viaggio insieme, scoprendo gli strumenti che renderanno il vostro percorso di apprendimento più produttivo e ordinato!

Struttura di un Progetto Python

La base di ogni progetto Python è una struttura di cartelle ben organizzata. Questo non è solo una questione di ordine, ma facilita la manutenzione e la collaborazione.

Cartella Principale

Create una cartella con il nome del vostro progetto. Questa conterrà tutti i file e le sottocartelle necessarie.

Sottocartelle Essenziali

All'interno della cartella principale, create sottocartelle come: "src" (per il codice sorgente), "tests" (per i test), "docs" (per la documentazione), e "data" (per i dati di esempio).

File di Configurazione

Aggiungete file come README.md per la descrizione, .gitignore per escludere file dal controllo versione, e requirements.txt per le dipendenze.

Una buona struttura di progetto vi permette di trovare rapidamente ciò che cercate e aiuta altri sviluppatori a comprendere l'organizzazione del vostro lavoro senza confusione.

Ambienti Virtuali: Cosa Sono e Perché Usarli

Cos'è un Ambiente Virtuale

Un ambiente virtuale è uno spazio isolato dove potete installare pacchetti Python senza interferire con altri progetti o con l'installazione principale di Python sul vostro sistema.

Vantaggi

Evita conflitti tra versioni diverse di librerie, mantiene il sistema pulito e permette di replicare facilmente l'ambiente su altre macchine.

Quando Usarlo

Praticamente sempre! Ogni progetto dovrebbe avere il proprio ambiente virtuale, anche quelli più semplici. È una buona abitudine che vi risparmierà molti problemi in futuro.

Gli ambienti virtuali sono fondamentali per mantenere l'ordine nel vostro ecosistema Python. Immaginate di lavorare contemporaneamente su due progetti che richiedono versioni diverse della stessa libreria: senza ambienti virtuali, questo sarebbe impossibile o estremamente complicato.

Creazione e Attivazione di un Ambiente Virtuale

Creazione dell'ambiente

Aprite il terminale nella cartella del progetto e digitate:

```
python -m venv nome_ambiente
```

dove "nome_ambiente" è il nome che volete dare (spesso si usa semplicemente "venv" o "env").

Attivazione su Windows

Nel prompt dei comandi:

```
nome_ambiente\Scripts\activate
```

o in PowerShell:

```
.\nome_ambiente\Scripts\Activate.ps1
```

Attivazione su macOS/Linux

Nel terminale:

```
source nome_ambiente/bin/activate
```

Verifica attivazione

Una volta attivato, vedrete il nome dell'ambiente tra parentesi all'inizio della riga di comando. Es:

```
(venv) C:\progetti\mioprogetto>
```

Ricordate che dovete attivare l'ambiente virtuale ogni volta che aprite un nuovo terminale o prompt dei comandi. L'attivazione modifica temporaneamente il PATH del sistema in modo che Python cerchi i pacchetti nell'ambiente virtuale anziché nell'installazione globale.

Installazione di Pacchetti nell'Ambiente Virtuale

Cercare i Pacchetti

Prima di installare, potete cercare informazioni sui pacchetti su **PyPI** (Python Package Index) all'indirizzo pypi.org o usando

```
pip search nome_pacchetto
```

(non più supportato nelle versioni recenti di pip).

Installare Pacchetti Singoli

Con l'ambiente attivato, usate:

```
pip install nome_pacchetto
```

Per esempio:

```
pip install requests
```

per installare la libreria requests.

Specificare Versioni

Potete richiedere una versione specifica:

```
pip install requests==2.28.1
```

o una versione minima:

```
pip install "requests>=2.20.0"
```

Verificare l'Installazione

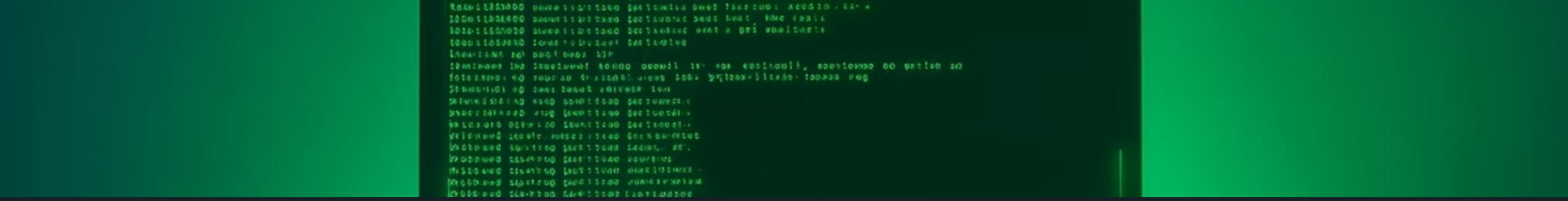
Per vedere tutti i pacchetti installati:

```
pip list
```

Per dettagli su un pacchetto specifico:

```
pip show nome_pacchetto
```

L'installazione di pacchetti nell'ambiente virtuale è uno dei principali vantaggi di questo approccio. Potete sperimentare liberamente senza preoccuparvi di danneggiare altri progetti o il sistema operativo.



Disinstallazione e Pulizia dell'Ambiente

Rimuovere Pacchetti Singoli

Per disinstallare un pacchetto specifico:

```
pip uninstall nome_pacchetto
```

Il comando chiederà conferma prima di procedere.

Pulizia Completa

Per rimuovere tutti i pacchetti, dovrete disinstallarli uno ad uno o, più drasticamente, eliminare l'intero ambiente virtuale e ricrearlo.

Ricreare l'Ambiente

Se l'ambiente è troppo "inquinato", potete disattivarlo con

```
deactivate
```

, eliminare la cartella dell'ambiente e crearne uno nuovo.

Mantenere pulito l'ambiente è importante, soprattutto nelle fasi iniziali di apprendimento. Pacchetti non necessari possono causare conflitti o problemi di dipendenze inaspettati. Ricordate che eliminare e ricreare un ambiente virtuale è un'operazione sicura e spesso più veloce che cercare di risolvere problemi complessi di dipendenze.

Salvataggio delle Dipendenze con pip freeze

Generare requirements.txt

Con l'ambiente attivato, eseguite:

```
pip freeze > requirements.txt
```

Questo comando salva l'elenco di tutti i pacchetti installati con le relative versioni.

Condividere con Altri

Il file può essere condiviso con altri sviluppatori per permettere loro di ricreare esattamente lo stesso ambiente di sviluppo.

Verificare il Contenuto

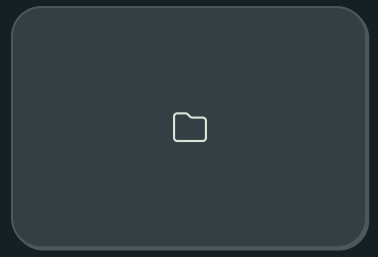
Aprite il file requirements.txt per verificare che contenga tutte le dipendenze necessarie. Potete modificarlo manualmente se necessario.

Versionare il File

Aggiungete il file requirements.txt al vostro sistema di controllo versione (es. Git) per tracciare le modifiche alle dipendenze nel tempo.

Il file requirements.txt è fondamentale per garantire la riproducibilità del vostro ambiente. Aggiornatelo ogni volta che installate o rimuovete pacchetti significativi. Per progetti più complessi, potrete esplorare strumenti più avanzati come Poetry o Pipenv, ma pip freeze è un ottimo punto di partenza.

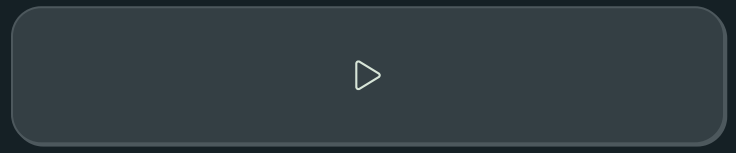
Ricreare l'Ambiente da requirements.txt



Creare nuovo ambiente virtuale

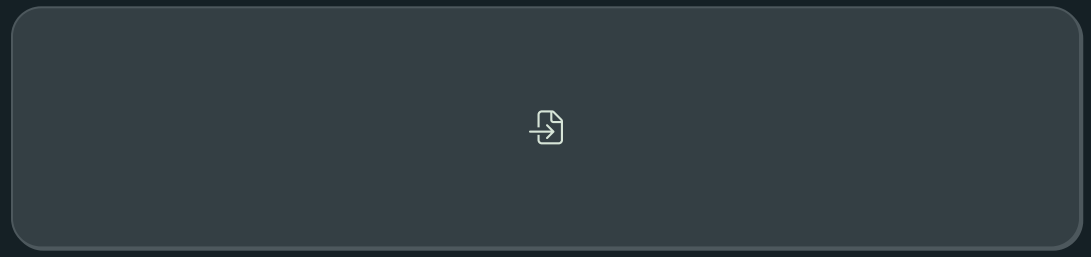
Create un nuovo ambiente virtuale con

```
python -m venv nuovo_ambiente
```



Attivare l'ambiente

Attivate l'ambiente secondo il vostro sistema operativo



Installare le dipendenze

Eseguite

```
pip install -r requirements.txt
```

La magia di requirements.txt sta nella sua semplicità: con un solo comando, potete ricostruire un ambiente identico a quello originale. Questo è particolarmente utile quando:

- Passate a un nuovo computer
- Condividete il vostro progetto con colleghi
- Dovete ricreare un ambiente dopo un problema
- Deployate la vostra applicazione in produzione

Fate attenzione ai potenziali problemi di compatibilità tra sistemi operativi diversi, specialmente con pacchetti che contengono estensioni compilate. In questi casi, potreste dover adattare manualmente alcune dipendenze.

Creare Script che Usano Pacchetti Esterni

Importare Correttamente

All'inizio del vostro script, importate i pacchetti necessari:

```
import requests
import pandas as pd
from datetime import datetime

# Resto del codice...
```

Raggruppate gli import per tipo: libreria standard, pacchetti di terze parti, moduli locali.

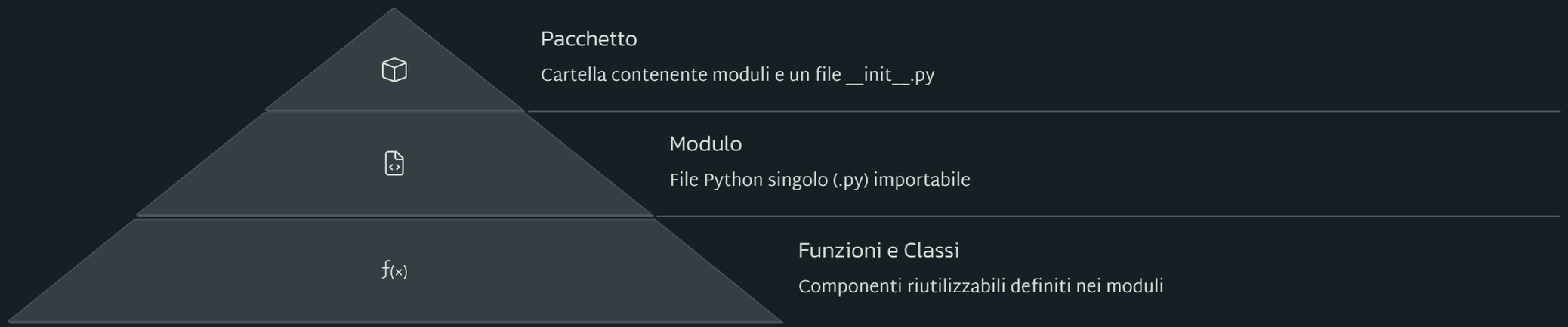
Esempio Pratico: Web Scraping

Ecco un esempio di script che utilizza pacchetti esterni per ottenere dati da un'API e analizzarli:

```
import requests
import pandas as pd
import matplotlib.pyplot as plt

# Ottenere dati da un'API
url = "https://api.coincap.io/v2/assets"
response = requests.get(url)
data = response.json()["data"]

# Convertire in DataFrame
df = pd.DataFrame(data)
top10 = df.sort_values("marketCapUsd",
                       ascending=False).head(10)

print(top10[["name", "priceUsd"]])

# Analisi e visualizzazione
# ...
```

Quando create script che utilizzano pacchetti esterni, consultate sempre la documentazione ufficiale per comprendere l'uso corretto delle API. Gestite anche possibili errori con blocchi try/except per rendere il vostro codice più robusto.

Struttura dei Moduli in un Progetto Python



Un progetto Python ben strutturato organizza il codice in moduli e pacchetti logici. Un modulo è semplicemente un file Python (.py) che può essere importato. Un pacchetto è una cartella contenente moduli e un file speciale chiamato __init__.py (anche vuoto nelle versioni recenti di Python).

Ecco un esempio di struttura per un'applicazione di analisi dati:

```
mio_progetto/
│
├── mio_pacchetto/
│   ├── __init__.py
│   ├── data_loader.py
│   ├── processor.py
│   └── visualizer.py
│
├── tests/
│   ├── __init__.py
│   ├── test_loader.py
│   └── test_processor.py
│
└── main.py
```

Questa organizzazione favorisce il riutilizzo del codice e semplifica la manutenzione a lungo termine.

Analisi di File CSV con Pandas

200+

Funzioni

Pandas offre centinaia di funzioni per manipolare, pulire e analizzare dati strutturati.

5M+

Righe

Può gestire facilmente dataset con milioni di righe, ottimizzando l'uso della memoria.

85%

Progetti

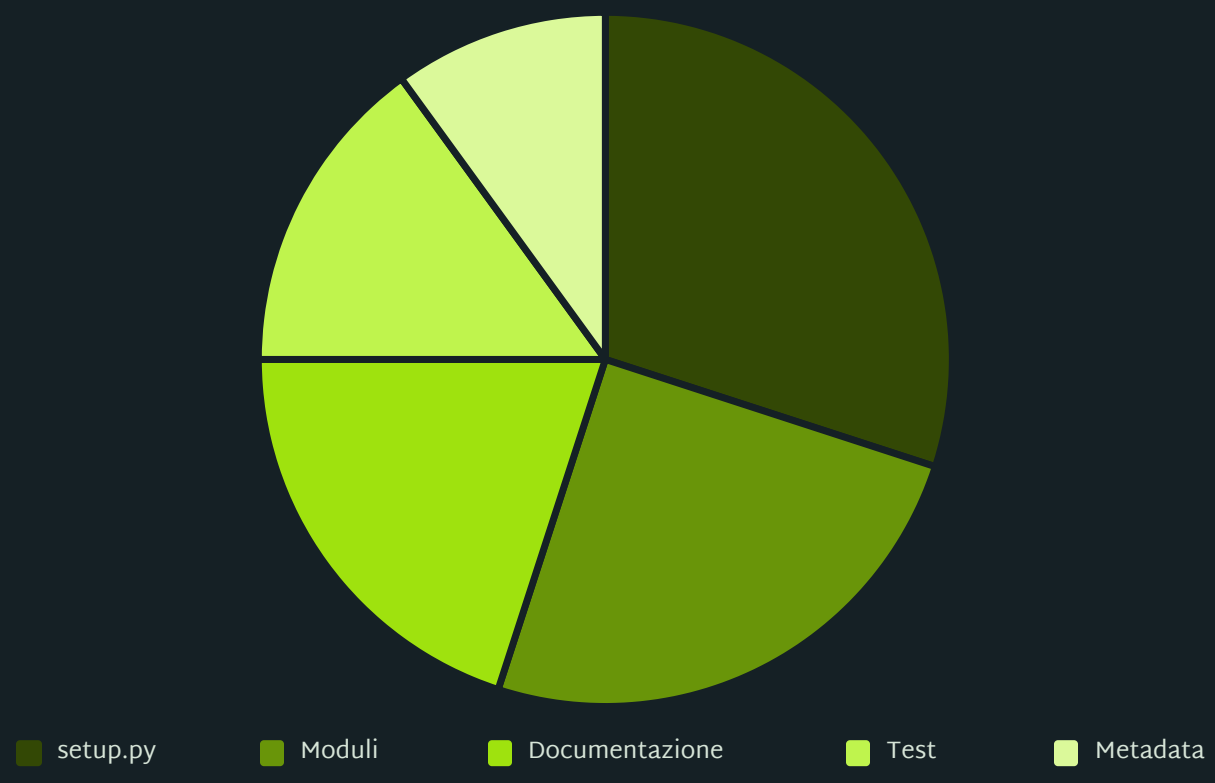
Percentuale approssimativa di progetti di data science che utilizzano Pandas come strumento principale.

Pandas è una libreria essenziale per l'analisi dei dati in Python. Ecco un esempio di come utilizzarla per analizzare un file CSV:

```
import pandas as pd
import matplotlib.pyplot as plt

# Caricamento dati
df = pd.read_csv("vendite.csv")

# Ispezione iniziale
print(df.head())  # Prime 5 righe
print(df.info())  # Informazioni sulle colonne
print(df.describe())  # Statistiche descrittive

# Analisi
vendite_mensili = df.groupby('mese')['importo'].sum()

# Visualizzazione
vendite_mensili.plot(kind='bar')
plt.title('Vendite Mensili')
plt.savefig('vendite_mensili.png')
plt.show()
```

Con poche righe di codice, potete caricare, pulire, analizzare e visualizzare dati da qualsiasi fonte strutturata. Questa capacità è ciò che rende Python così potente per l'analisi dei dati.

Creazione di un Semplice Pacchetto Python



Creare un pacchetto Python è più semplice di quanto si possa pensare. La struttura minima richiede solo alcuni file chiave:

```
mio_pacchetto/
├── setup.py
├── README.md
└── mio_pacchetto/
    ├── __init__.py
    └── modulo.py
```

Il file setup.py è il cuore del pacchetto e contiene metadati e configurazioni:

```
from setuptools import setup, find_packages

setup(
    name="mio_pacchetto",
    version="0.1.0",
    author="Il Vostro Nome",
    author_email="email@esempio.com",
    description="Breve descrizione del pacchetto",
    packages=find_packages(),
    install_requires=[
        "requests>=2.25.0",
        "pandas>=1.2.0",
    ],
)
```

Una volta creato il pacchetto, potete installarlo in modalità sviluppo con

```
pip install -e .
```

eseguito dalla cartella che contiene setup.py.

Debug dell'Ambiente Virtuale

Identificare il Problema

Verificate se il problema è legato all'ambiente virtuale o al codice stesso. Errori comuni includono percorsi Python errati, dipendenze mancanti o versioni incompatibili.

Raccogliere Informazioni

Usate comandi come

```
which python
```

(Linux/Mac) o

```
where python
```

(Windows) per verificare quale Python state usando.

```
pip list
```

mostra i pacchetti installati.

Risolvere i Problemi

Per problemi comuni: reinstallate i pacchetti, verificate i percorsi nell'ambiente, assicuratevi che l'ambiente sia attivato, o in casi estremi, ricreatelo da zero.

Verificare la Soluzione

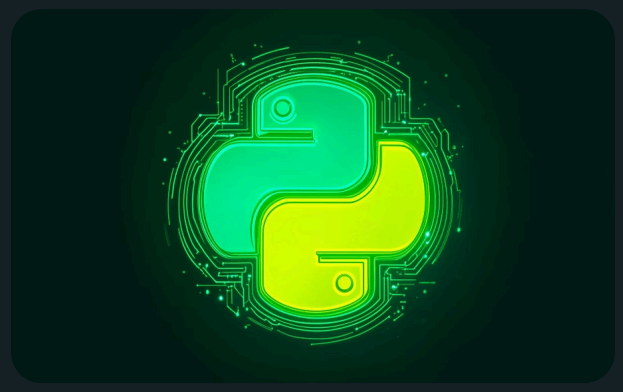
Create uno script di test minimo per verificare che l'ambiente funzioni correttamente dopo le modifiche.

Un problema comune è quello del "PATH di Python". Quando attivate un ambiente virtuale, il sistema dovrebbe usare il Python dell'ambiente, non quello globale. Verificate con:

```
# In Linux/Mac
which python
which pip

# In Windows
where python
where pip
```

Se questi comandi non puntano alla cartella del vostro ambiente virtuale, potrebbe esserci un problema con l'attivazione o con le variabili d'ambiente.

Condivisione dell'Ambiente e Controllo Versione



Git per il Controllo Versione

Git è lo strumento standard per il controllo versione. Create un repository con

```
git init
```

nella cartella del progetto, aggiungete i file con

```
git add .
```

e committate con

```
git commit -m "Messaggio"
```

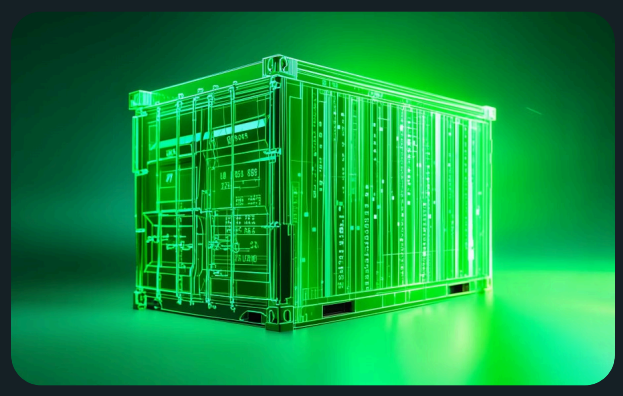
.

Condivisione con requirements.txt

Il file requirements.txt è il metodo più semplice per condividere dipendenze. Ricordatevi di aggiornarlo ogni volta che modificate l'ambiente con

```
pip freeze > requirements.txt
```

.



Containerizzazione Avanzata

Per progetti più complessi, considerate Docker per impacchettare non solo le dipendenze Python, ma l'intero ambiente di esecuzione, garantendo che funzioni allo stesso modo ovunque.

La combinazione di ambienti virtuali, requirements.txt e Git è la base per una collaborazione efficace nei progetti Python. Includete sempre nel vostro .gitignore la cartella dell'ambiente virtuale (es. /venv/), poiché ogni sviluppatore dovrebbe creare il proprio ambiente locale basato sul requirements.txt, non condividere quello esistente.

Ricordate che la documentazione è fondamentale: includete sempre istruzioni chiare su come configurare l'ambiente nel README.md del vostro progetto, in modo che chiunque possa iniziare a lavorare rapidamente sul vostro codice.