PBG

Python: Dalla Preistoria agli Ambienti Virtuali

Ovvero: come NON trasformare il vostro PC in un campo di battaglia di dipendenze

💻 III Liceo Scientifico Biella - Scienze Applicate

Python Biella Group

📜 C'era una volta... Python

1991: Guido van Rossum inventa Python durante le vacanze di Natale
(Mentre i vostri genitori giocavano a Snake sul Nokia, lui creava il serpente definitivo)

Obiettivo: Un linguaggio semplice, leggibile, potente
Ispirazione del nome: Monty Python's Flying Circus
(No, non il rettile. Delusione per tutti)

Fun fact: Guido voleva solo passare il tempo. Ha creato uno dei linguaggi più usati al mondo.
Casual.

👥 Chi usa Python oggi?

- **Data Scientists**: "Pandas è la mia vita" 🐼
- **Web Developers**: Django e Flask per il web che spacca
- **AI/ML Engineers**: TensorFlow, PyTorch, scikit-learn
- **DevOps**: Automazione di tutto l'automatizzabile
- **Hacker etici**: Penetration testing e security
- **Voi**: Futuri dominatori del mondo tech 🚀

Anche: NASA, Google, Netflix, Spotify, Instagram...

Praticamente, se ha successo, probabilmente usa Python

🎯 SFIDA #1: Trova l'intruso

Quale di questi NON è un motivo valido per usare Python?

A) Sintassi pulita e leggibile

B) Enorme ecosistema di librerie

C) Performance da Formula 1

D) Eccellente per prototipazione rapida

Spoiler: Python è fantastico, ma veloce come un F1? Ecco, no. 🐌

🔢 Le Versioni di Python: Una Saga Epica

Python 1.x (1994): L'inizio. Lambda, map, filter. Roba seria.

Python 2.x (2000-2020): L'era d'oro... poi diventata legacy

- `print` senza parentesi: `print "ciao"` *(che caos)*
- Ufficialmente morto il 1 gennaio 2020 💀

Python 3.x (2008-oggi): Il presente e il futuro

- `print("Finalmente con le parentesi!")`
- NON retrocompatibile → trauma collettivo mondiale

🚀 Python 3: Le Versioni che Contano

- **Python 3.6** (2016): f-strings! `f"Ciao {nome}"`
- **Python 3.8** (2019): Walrus operator `:=` (perché no?)
- **Python 3.10** (2021): Pattern matching strutturale
- **Python 3.11** (2022): +25% di velocità! 🏃
- **Python 3.12** (2023): Ancora più veloce
- **Python 3.13** (2024): JIT compiler sperimentale

Regola d'oro: Usate SEMPRE Python 3.9+

(3.8 e precedenti sono tipo il MySpace dei linguaggi)

PEP: Python Enhancement Proposals

Cos'è un PEP? Descrizione di una nuove feature per Python

Il più famoso: **PEP 8** - Style Guide for Python Code

- Indentazione a 4 spazi (NO tab, fight me)
- Max 79 caratteri per riga (ok boomer)
- Snake_case per funzioni e variabili

Altri PEP iconici:

- **PEP 20**: The Zen of Python (`import this`)
- **PEP 484**: Type Hints
- **PEP 572**: Il controverso Walrus Operator

🧘 The Zen of Python (PEP 20)

```
>>> import this
```

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- **Readability counts** ⭐

Fondamentalmente: scrivete codice come se dovesse leggerlo uno psicopatico che sa dove abitate.
E quello psicopatico siete voi tra 6 mesi.

🎯 SFIDA #2: PEP Quiz

Quale tra questi è un VERO PEP?

A) PEP 404 - Python Not Found

B) PEP 666 - Summoning Daemons in Python

C) PEP 3000 - Python 3000

D) PEP 8000 - Time Travel in Python

Hint: Python ha un senso dell'umorismo, ma è anche serio

🔄 Come Python Evolve

1. **Qualcuno ha un'idea** → Scrive un PEP
2. **Discussione comunitaria** → Flame wars incluse 🔥
3. **Core Developers Review** → Il vero giudizio
4. **BDFL/Steering Council** decide → Guido era il "Benevolent Dictator For Life" (ora abbiamo un consiglio)
5. **Implementazione** → Codice vero
6. **Release** → Party time! 🎉

Ciclo di rilascio: Una major release all'anno (ottobre)

Supporto: ~5 anni per versione

🌍 Il Problema delle Versioni Multiple

Scenario realistico:

- Progetto A richiede Django 3.2
- Progetto B richiede Django 4.0
- Entrambi sullo stesso sistema
- Django si installa globalmente

Risultato: 💣 KABOOM 💣

Uno dei due progetti esplode. O entrambi. È la legge di Murphy.

😱 L'Inferno della Dependency Hell

Cosa succede senza ambienti virtuali?

```
pip install package_figo
```

- Si installa **globalmente** sul sistema
- Sovrascrive versioni esistenti
- Crea conflitti con altri progetti
- Il vostro sistema diventa un cimitero di dipendenze zombie

Il debug: 6 ore per capire perché tutto si è rotto

La soluzione: Reinstallare Python e piangere 😭

🎯 SFIDA #3: Disaster Scenario

Avete installato 47 pacchetti globalmente. Ora dovete:

- Deployare il progetto su un server
- Far funzionare il codice sul PC del vostro compagno
- Ricordare quali pacchetti servono DAVVERO

Tempo stimato: ∞
Probabilità di successo: 0.0001%
Livello di frustrazione: 🔥🔥🔥🔥🔥

Soluzione: Ambienti virtuali. Always.

Ambienti Virtuali: I Vostri Salvatori

Cosa sono?

Cartelle isolate con:

- Propria copia di Python
- Propri pacchetti installati
- Zero interferenze con il sistema

Vantaggi:

✅ Progetti indipendenti
✅ Versioni diverse di librerie
✅ Facile condivisione (via `requirements.txt`)
✅ Sistema pulito e organizzato
✅ Sonni tranquilli 😴

venv: Il Classico Intramontabile

Incluso in Python 3.3+ (non serve installare nulla!)

Creazione (Windows):

```
python -m venv nome_ambiente
nome_ambiente\Scripts\activate
```

Creazione (macOS/Linux):

```
python3 -m venv nome_ambiente
source nome_ambiente/bin/activate
```

Prompt attivato: `(nome_ambiente) C:\Users\...`

🔧 venv: Workflow Completo

```
# 1. Crea ambiente
python -m venv myproject_env

# 2. Attiva
myproject_env\Scripts\activate  # Windows
source myproject_env/bin/activate  # Mac/Linux

# 3. Installa roba
pip install numpy pandas matplotlib

# 4. Salva dipendenze
pip freeze > requirements.txt

# 5. Disattiva quando finito
deactivate
```

📝 requirements.txt: Il Manifesto

File che lista tutte le dipendenze:

```
numpy==1.24.3
pandas==2.0.2
matplotlib==3.7.1
scikit-learn==1.2.2
```

Installare tutto su un altro PC:

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

È tipo la lista della spesa, ma per nerd. 🛒

🎯 SFIDA #4: venv Master

Qual è l'ordine CORRETTO delle operazioni?

A) Installa pacchetti → Crea venv → Attiva → Codifica

B) Crea venv → Installa pacchetti → Attiva → Codifica

C) Crea venv → Attiva → Installa pacchetti → Codifica

D) Attiva → Crea venv → Codifica → Installa pacchetti

Hint: La logica esiste, usatela 🧠

⚡ uv: Il Nuovo Ragazzo Figo

Cosa è uv?

Tool moderno scritto in Rust (quindi velocissimo 🚀)

- Gestione ambienti virtuali
- Gestione pacchetti
- **10-100x più veloce di pip**

Installazione:

```
# Windows (PowerShell)
irm https://astral.sh/uv/install.ps1 | iex

# macOS/Linux
curl -LsSf https://astral.sh/uv/install.sh | sh
```

🚀 uv: Velocità Supersonica

Creare ambiente con uv:

```
# Crea e attiva in un colpo
uv venv

# Attiva (come venv)
.venv\Scripts\activate  # Windows
source .venv/bin/activate  # Mac/Linux
```

Installare pacchetti:

```
uv pip install numpy pandas matplotlib
```

Differenza: Mentre pip fa colazione, uv ha già finito ☕→⚡

⚔️ venv vs uv: Il Confronto

	🐌	🚀
Installazione	✅ Built-in	❌ Da installare
	✅	⚠️
Semplicità	✅ Semplice	✅ Semplice
	😐	😎

Verità: Entrambi fanno il lavoro. uv è più veloce, venv è già lì.

💡 Best Practices da Ninja

1. **UN ambiente virtuale per progetto** (sempre!)
2. **Mai committare la cartella dell'ambiente** (.gitignore!)
3. **Aggiorna requirements.txt** quando aggiungi pacchetti
4. **Usa Python 3.10+** per nuovi progetti
5. **Attiva l'ambiente PRIMA di installare** (quante volte devo ripeterlo?)
6. **Nome standard**: `venv`, `.venv`, o `nome_progetto_env`

Seguite queste regole e i vostri progetti ringrazieranno 🙏

🎯 SFIDA FINALE: Scenario Reale

Dovete:

1. Creare un progetto di Data Science
2. Usare NumPy 1.24, Pandas 2.0, Matplotlib 3.7
3. Condividere con il team
4. Non distruggere il vostro sistema

Quale approccio usate?

A) Installo tutto globalmente (YOLO)
B) Uso un ambiente virtuale come un pro
C) Chiedo all'IA di farlo per me
D) Abbandono e divento influencer

✅ La Soluzione (Ovviamente B)

```
# Step by step per vincere
mkdir progetto_data_science
cd progetto_data_science

# Opzione 1: venv
python -m venv venv
venv\Scripts\activate
pip install numpy==1.24 pandas==2.0 matplotlib==3.7
pip freeze > requirements.txt

# Opzione 2: uv (se sei cool)
uv venv
.venv\Scripts\activate
uv pip install numpy==1.24 pandas==2.0 matplotlib==3.7
uv pip freeze > requirements.txt
```

🎓 Cosa Abbiamo Imparato

- ✅ Python: da hobby a dominatore del mondo tech
- ✅ PEP: come Python evolve democraticamente
- ✅ Versioni: 3.10+ o morte
- ✅ **Ambienti virtuali**: NON sono opzionali
- ✅ venv: affidabile e sempre disponibile
- ✅ uv: il futuro ad alta velocità
- ✅ Dependency Hell: si previene, non si cura

Motto: "*Un progetto, un ambiente. Sempre.*"

🚀 Challenge per Casa

Missione: Create 3 ambienti virtuali diversi

1. **data_science_env**: NumPy, Pandas, Matplotlib
2. **web_dev_env**: Flask, Requests
3. **ml_env**: scikit-learn, TensorFlow

Obiettivo: Capire l'isolamento

Bonus: Provate sia con venv che con uv

Super Bonus: Cronometrate la differenza di velocità

Chi finisce per primo vince... la soddisfazione personale 🏆

Risorse Utili

- **Documentazione Python**: docs.python.org
- **PEP Index**: peps.python.org
- **Real Python**: realpython.com (tutorial fantastici)
- **uv Documentation**: docs.astral.sh/uv
- **Python Package Index**: pypi.org

Consiglio saggio: Leggete i PEP importanti. Seriamente. Vi faranno sembrare intelligenti.

Domande?

Remember:

- Gli ambienti virtuali sono vostri amici 🤝
- pip install != pip install globale
- Quando dubitate, create un nuovo venv
- Python è fantastico, ma l'isolamento lo è ancora di più

Prossima lezione: Faremo esplodere cose con NumPy! 💥

"With great power comes great responsibility"
- Uncle Ben (ma valeva anche per Python)

🎉 Grazie!

Ora andate e popolate il mondo di ambienti virtuali!

P.S.: Le risposte alle sfide le trovate pensando. Sì, dovete usare il cervello. Lo so, che fatica. 🧠

Extra credit: Chi installa qualcosa globalmente questa settimana paga le pizze 🍕