



# DelphiDay

italian conference

# DELPHI + PYTHON

Il “dinamico duo” dei linguaggi di programmazione!



# MARCO BREVEGLIERI

Software Developer @ABLS Team



Homepage  
<https://www.breveglieri.it>



Blog tecnico  
<https://www.compilaquindiva.com>



Delphi Podcast  
<https://www.delhipodcast.com>



Canale Twitch  
<https://twitch.tv/compilaquindiva>

Tutti gli altri link: <https://linktr.ee/marco.breveglieri>



20 Novembre 2024  
Padova





# AGENDA

- **Introduzione a Python**
  - Panoramica del linguaggio e della sintassi
  - Confronto con Delphi e ambiti di complemento
  - Sintassi, ecosistema e strumenti di sviluppo
- **Esploriamo *Python4Delphi***
  - Che cos'è la libreria *Python4Delphi*
  - Panoramica e utilizzo dei componenti
  - Passaggio di dati da/a Python
- **Dallo scripting alle applicazioni reali**
  - Uso dei Python Package più comuni
  - Add-on per l'uso di package e librerie
  - Panoramica di *Delphi4Python*
- **Wrap-up!**
  - Risorse e approfondimenti
  - Q & A



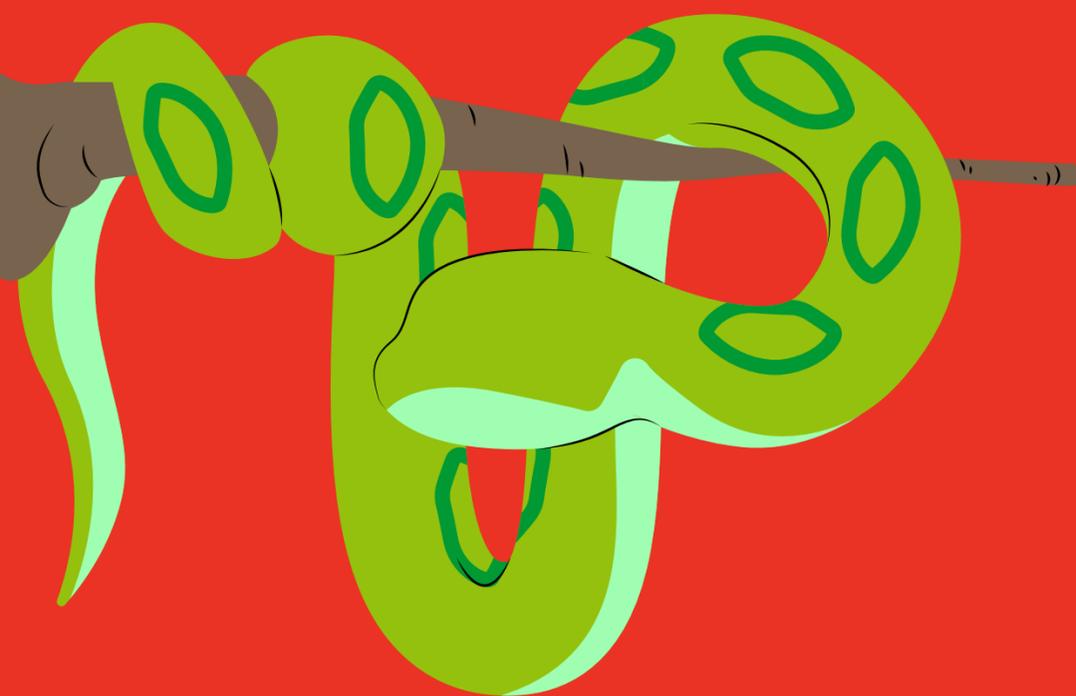


Python  Delphi

1



# Python (visto da un Delphista)



- Sintassi immediata, chiara e leggibile
- Adatto a una vasta gamma di applicazioni
- Librerie e framework per tutti gli usi
- Lo “standard de facto” per il Data Science
- Ampiamente utilizzato nell’ambito AI & ML
- Efficace per automazione di task e scripting
- Cross-platform e cross-device, ovvio!



# Delphi ~~vs~~ Python

- Non esiste un tool adatto a tutti gli scopi
- Gli sviluppatori devono avere più tool nella propria cassetta degli attrezzi
- Il gioco è trovare il tool giusto per assolvere uno specifico compito o raggiungere l'obiettivo
- Avere strumenti per ogni specifica esigenza è cosa del tutto normale, anzi è consigliato!





# Perché dovrei conoscere Python?

- Popolarità alta e crescente del linguaggio
- Linguaggio scelto per Data Analytics, Machine Learning e Artificial Intelligence
- Sta sostituendo Java negli istituti accademici
- Pletora di librerie e package disponibili e pronti all'uso
- Percepito come facile da usare e orientato alla produttività
- Possiede molti punti di forza in comune e complementari con Delphi





# Una sinergia perfetta!



- Accesso alle librerie Python da applicazioni Delphi
- Uso di Python come linguaggio di scripting
- Rendere accessibile codice scritto in Delphi da applicazioni e script in linguaggio Python
- Affiancare funzionalità RAD e librerie per lo sviluppo di GUI al runtime di Python
- Combinare le potenzialità di entrambi i linguaggi di programmazione



# Un rapido confronto

	Delphi/Pascal	Python
Maturity	✓ (1995/1970!)	✓ (1989)
Object orientation	✓	✓
Multi-platform	✓	✓
Verbosity	High (begin end)	Low (indentation based)
REPL	No	Yes
Typing	Strong static typing	Dynamic (duck) typing
Memory management	Manual	Reference counting
Compiled	✓	bytecode
Performance	👍	👎
Multi-threading	👍	👎
RAD	👍	👎



# Introduzione a Python



# 2



# Il linguaggio Python

---

- Sintassi semplice e leggibile
- Relativamente facile da imparare
- Utile in molti ambiti
  - Scienza e analisi dati
  - Sviluppo Web
  - Programmazione GUI\*
- Libreria standard molto ampia
- Pacchetti aggiuntivi infiniti

*(\*) ma esiste anche l'opzione Delphi. 😊*



# Versioni di Python

- Python 3: nuova “minor version” (es. v3.13.0) rilasciata ogni ottobre
- Python 2: il supporto è terminato nel 2019
  - Ancora usato dal 10% degli sviluppatori



# Esempio di codice

```
# this is a comment  
  
a = 3  
b = 4  
print(a * b)  
  
if a * b > 10:  
    print('greater')
```



# Installazione su Windows /1

Scarica la versione aggiornata da:  
<https://python.org>

Durante l'installazione, ricordarsi di **spuntare l'opzione**  
*"Add Python 3.x to PATH"*

Visualizza la versione corrente con il comando  
**python --version**

Serve aiuto?  
Esegui: **help(...)**



# Installazione su Windows /2

Il pacchetto di installazione include:

- Il *runtime* di Python
- La console interattiva (REPL)
- Un ambiente di sviluppo rudimentale: IDLE
- Un *package manager* integrato: PIP



# Console interattiva di Python

---

Come eseguire codice in Python:

- Programma o script scritto in un file (o collezione di file) da eseguire a comando (per applicazioni a riga di comando, app Web, GUI, ecc.)
- Codice digitato “al volo” all’interno della Console REPL (o in un Notebook) riga per riga, per sperimentazioni, analisi di dati, calcoli veloci, ecc.)



# Proviamo!





# Un editor più evoluto?

---

Per la scrittura di programmi strutturati, possiamo usare editor o IDE completi e più evoluti:

- **PyScripter** (scritto in Delphi con *Python4Delphi*)
- **PyCharm** (*IDE* completo, prodotto da JetBrains)
  - Edizione “Community”
  - Edizione “Professional”
- **Visual Studio Code** (*editor*)
  - Open-source, con estensioni dedicate



# Il nostro primo programma

```
print("What is your name?")  
name = input()  
print("Nice to meet you, " + name)
```

```
python hello.py
```



# Variabili /1

Queste le regole e convenzioni per i nomi delle variabili:

- Sono scritti in minuscolo (*lower case*)
- Le parole sono separate da underscore (`_`)
- Posso essere formate da lettere, numeri e underscore (`_`)

```
birth_year = 1970
current_year = 2020
age = current_year - birth_year
```



# Variabili /2

Le variabili possono essere riassegnate.

I tipi delle variabili possono cambiare dinamicamente, in base al valore loro assegnato.

NOTA: esiste comunque un minimo supporto “type safe”... non è come JavaScript.

```
name = "John"  
name = "Jane"  
a = 3  
a = a + 1
```



# Tipi e conversioni

- E' possibile determinare il tipo di una variabile tramite l'uso di *type*
- Gli oggetti si possono convertire tramite le funzioni *int()*, *float()*, *str()*, *bool()*, ...

```
a = 4 / 2
type(a)

pi = 3.1415
pi_int = int(pi)
message = "Pi is approximately " + str(pi_int)
```



# Controllo del flusso

Il codice viene strutturato tramite indentazione (con 4 spazi in genere) e iniziando blocchi con il carattere “:”

```
if ... else ...  
while  
for ... in ...  
for ... in range(...)  
  
def average(a, b):  
    m = (a + b) / 2  
    return m
```



# Funzioni

---

Una funzione è una “subroutine” che può eseguire uno o più compiti specifici.

Esempi di funzioni predefinite:

- *len()* determina la lunghezza di una stringa, di una lista, ...
- *id()* determina l'identificativo interno di un oggetto
- *type()* restituisce il tipo di un oggetto
- *print()* scrive output sul terminale

...



# Parametri e valore di ritorno

---

Una funzione può ricevere zero, uno o più parametri e produrre in uscita un valore di ritorno (*return value*).

Ad esempio:

*len()* prende una stringa e restituisce la lunghezza come intero

*print()* prende valori eterogenei e non restituisce un valore (\*)

(\*) *quantomeno, non esplicitamente*



# Metodi

Un metodo è una funzione che appartiene a uno specifico tipo di oggetto.

Esempi di metodi per il tipo “stringa”:

```
first_name.upper()  
first_name.count("a")  
first_name.replace("a", "@")
```



# Dizionari

I dizionari sono strutture che contengono elementi con nome (chiavi) con i relativi valori associati.

Per recuperare e impostare gli elementi:

```
person = {  
    "first_name": "John",  
    "last_name": "Doe",  
    "nationality": "Canada",  
    "birth_year": 1980  
}
```

```
person["first_name"]  
person["first_name"] = "Jane"
```



# Liste /1

Una lista rappresenta una sequenza di oggetti: E' possibile recuperare elementi in questo modo:

```
primes = [2, 3, 5, 7, 11]

users = ["Alice", "Bob", "Charlie"]

products = [
    {"name": "iPhone 12", "price": 949},
    {"name": "Fairphone", "price": 419},
    {"name": "Pixel 5", "price": 799}
]
```

```
users = ["Alice", "Bob", "Charlie"]

users[0]
users[1]
users[-1] # last element
```



# Liste /2

---

Sovrascrivere un elemento:

```
users[0] = "Andrew"
```

Rimozione dell'ultimo elemento:

```
users.pop()
```

Accodare un elemento:

```
users.append("Dora")
```

Rimozione in base all'indice:

```
users.pop(0)
```

Lunghezza della lista:

```
len(users)
```



# Tuple

---

Le tuple consentono di memorizzare più valori in una unica variabile.

```
date = (1973, 10, 23)
```

*NOTA: gli elementi delle tuple sono immutabili!*



# Tuple vs dizionari

- Le tuple hanno un ambito di utilizzo simile ai dizionari.
- Ogni elemento della tupla ha un significato preciso.

```
point_dict = {"x": 2, "y": 4}
point_tuple = (2, 4)

date_dict = {
    "year": 1973,
    "month": 10,
    "day": 23
}
date_tuple = (1973, 10, 23)
```



# Tuple vs liste

- Le tuple supportano l'accesso "posizionale" ai valori
- A differenza delle liste, le tuple sono immutabili, come predetto

```
date_tuple[0] # 1973
```

```
len(date_tuple) # 3
```



# Proviamo!





# Oggetti e riferimenti /1

Quiz: quale sarà il valore stampato a video al termine dell'esecuzione di questo programma? 🕵️

```
a = [1, 2, 3]
b = a
b.append(4)
print(a)
```

```
[1, 2, 3, 4]
```



# Oggetti e riferimenti /2

Quando si trattano oggetti, l'assegnazione tra variabili riguarda il riferimento (*reference*) agli stessi.

Stiamo solo attribuendo una nuova “etichetta” all’oggetto.

L’oggetto è pertanto sempre lo stesso.





# Oggetti e riferimenti /3

Se il valore originale deve rimanere intatto, allora è necessario creare una copia dell'oggetto o derivarne uno nuovo da esso.

```
a = [1, 2, 3]
# creating a new copy
b = a.copy()
# modifying b
b.append(4)
```

```
a = [1, 2, 3]
# creating a new object b
b = a + [4]
```

NOTA:

i valori semplici sono immutabili!



# Errori e traceback

Alcune funzioni possono generare un errore, visualizzando

- il punto in cui si è verificato il problema
- una descrizione significativa dell'errore.

```
Traceback (most recent call last):  
  File "xyz.py", line 1, in <module>  
    open("foo.txt")  
FileNotFoundError: [Errno 2] No such file or directory: 'foo.txt'
```



# Funzioni “built-in” e standard library

- **Built-in**: funzioni e oggetti usati frequentemente e sempre disponibili
  - *print(), input(), len(), max(), min(), open(), range(), round(), sorted(), sum(), type()*
- **Standard Library**: raccolta di moduli e pacchetti aggiuntivi che possono essere importati

```
import math
print(math.floor(3.6))
```



# Moduli standard

---

Alcuni dei moduli Python della libreria standard più utilizzati:

- **pprint** (*pretty printing*)
- **random**
- **math**
- **datetime**
- **os** (*sistema operativo, file system*)
- **urllib.request** (*richieste HTTP*)
- **webbrowser** (*semplice controller per browser web*)

Per convenzione, i moduli vanno importati all'inizio.



# Proviamo!





# PIP and PyPI

---

E' possibile usare librerie aggiuntive con Python usando gli appositi tool:

- **PyPI** (*Python Package Index*): repository ufficiale per download e installazione di package Python ➡ <https://pypi.org>
- **PIP**: Package Manager per Python

E' possibile importare come moduli anche file Python locali.



# Esploriamo *Python4Delphi*



# 3

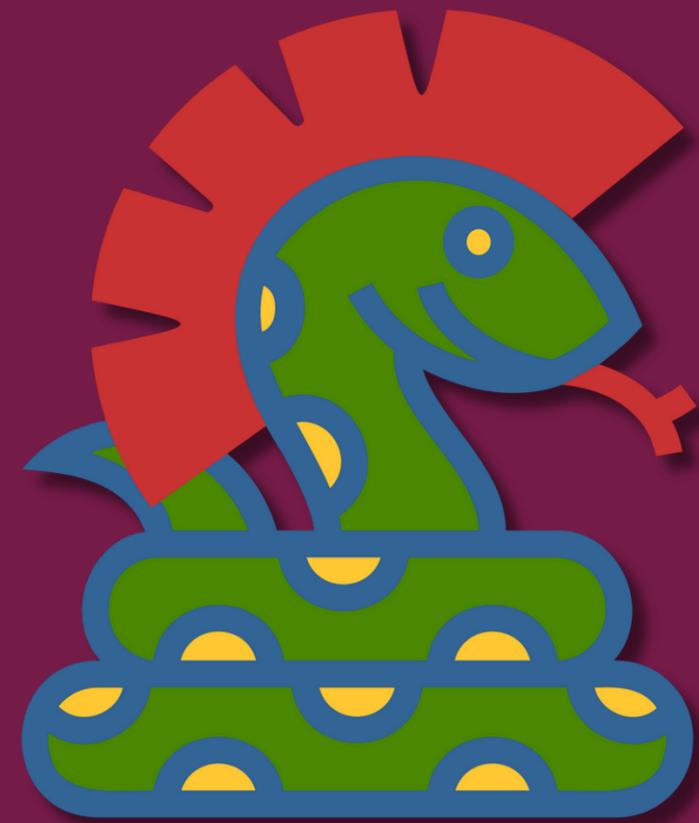


# Python for Delphi (P4D)

Python4Delphi è un set di componenti che permettono di interfacciare applicazioni *Delphi* e *Lazarus* (FPC) con il runtime di *Python*.

Consente di

- eseguire script scritti in linguaggio Python
- creare moduli e tipi in ambiente Python
- creare estensioni (DLL) per Python
- sfruttare un'API per interfacciare Delphi con Python (e viceversa)





# Caratteristiche di P4D

- Accesso a basso livello alle funzioni dell'API del runtime di Python
- Interazione bidirezionale con l'ambiente
- Accesso a oggetti Python tramite variabili di tipo Variant in Delphi in modo dinamico
- “Wrapping” di oggetti Delphi per esporne l'uso all'interno di script in Python
- Creazione di “extension module” (DLL) per Python contenenti classi e moduli, scritti in Delphi





# Installazione di P4D /1

---

- Download dei sorgenti dal repo su GitHub (<https://github.com/pyscripter/python4delphi>)
  - “Clone” via Git per ricevere gli update
- Aprire il gruppo di progetti “P4DComponentSuite” e fare *Build All*
- Installare i package designtime
- Configurare il *Library Path* nelle opzioni

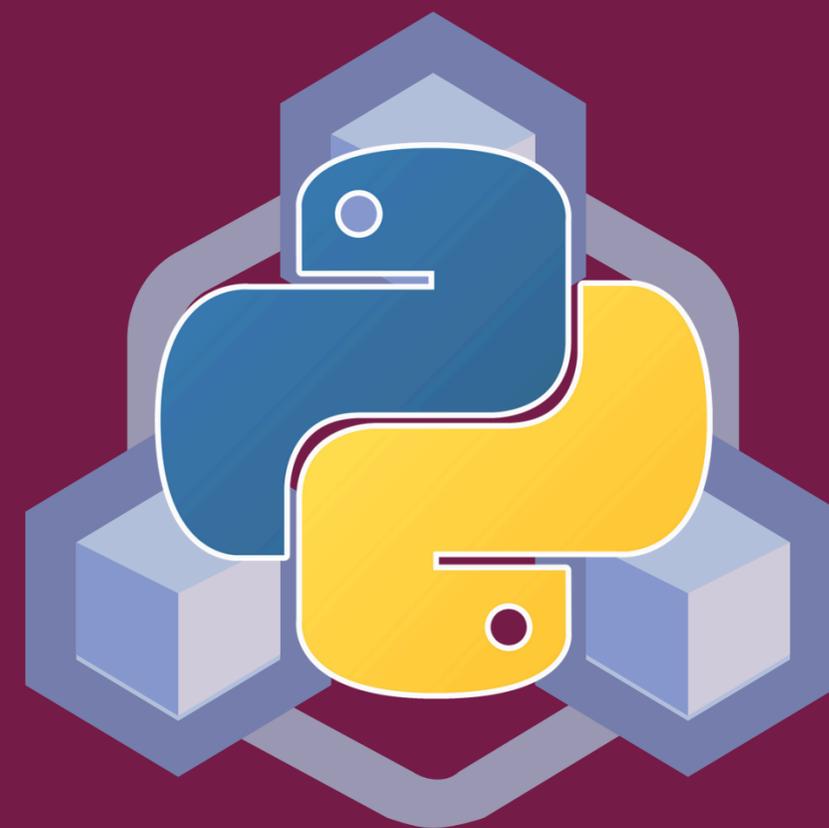




# Le Unit della libreria

Le unit principali della libreria *Python4Delphi* sono:

- *PythonEngine.pas*: collezione di routine di bassi livello per interagire con il runtime di Python (creazione di tipi, allocazione memoria, ecc.)
- *VarPyth.pas*: funzioni di alto livello per l'accesso facilitato a oggetti Python in Delphi tramite i custom variant.
- *WrapDelphi.pas*: classi che semplificano la manipolazione di GUI e altri oggetti Delphi all'interno di script Python.





# Componenti di P4D

Componente	Funzionalità
<b>TPythonEngine</b>	<i>Carica e connette il runtime di Python. Fornisce accesso a basso livello.</i>
<b>TPythonModule</b>	<i>Crea un modulo Python in Delphi e lo rende accessibile all'ambiente di Python.</i>
<b>TPythonType</b>	<i>Crea un tipo Python (classe) in Delphi.</i>
<b>TPythonInputOutput</b>	<i>Si aggancia all'I/O dell'interprete Python.</i>
<b>TPythonGUIInputOutput</b>	<i>Connette l'I/O dell'interprete Python a controlli della GUI.</i>
<b>TPyDelphiWrapper</b>	<i>Rende classi e oggetti Delphi accessibili al runtime di Python.</i>



# Proviamo!





# TPythonEngine: panoramica

- Permette di inizializzare Python caricando la DLL
- Include routine di alto livello per dialogare con il runtime di Python
- E' una classe Singleton: una sola per applicazione (si può usare *GetPythonEngine()* per recuperare l'istanza)



NOTA: attenzione alla “bitness” (32-bit oppure  64-bit) in fase di build!



# TPythonEngine: proprietà

- **AutoLoad**: carica la DLL di Python in automatico
- **AutoUnload**: come sopra, ma per scaricare la DLL
- **DllName**: contiene il nome della DLL di Python da caricare (default: ultima versione)
- **DllPath**: contiene la locazione fisica della DLL
- **InitScript**: lo script da eseguire in fase di inizializzazione
- **IO**: associabile ai componenti per l'I/O su componenti visuali.
- **RegVersion**: contiene la versione di Python rilevata come “registrata” nel sistema
- **UseLastKnownVersion**: usa automaticamente l'ultima versione conosciuta di Python





# TPythonEngine: metodi principali

- **SetPythonHome()**: imposta la directory home dell'ambiente virtuale di Python
- **LoadDll()**: effettua il caricamento della DLL del runtime
- **ExecString()**: esegue una singola riga di script
- **ExecStrings()**: esegue uno script di più righe
- **EvalStrings()**: esegue uno script e ottiene il valore restituito dall'espressione
- **PyObjectAsString()**: converte un oggetto Python qualsiasi nella sua rappresentazione testuale
- **Py\_INCREF()**: incrementa il "reference count"
- **Py\_DECREF()**: decrementa il "reference count"





# TPythonGUIInputOutput

Fornisce un canale di comunicazione per instradare output e input dello script in esecuzione ai controlli della GUI, sia VCL sia FMX.

- **MaxLineLength**: massimo numero di caratteri consentiti per singola riga
- **MaxLines**: numero massimo di righe dello script che possono essere eseguite
- **Output**: consente di associare un TMemo per visualizzare il testo in uscita
- **RawOutput**: usa un formato “grezzo” per il testo
- **UnicodeIO**: regola l’uso di Unicode come encoding per l’input e l’output





# Proviamo!





# TPythonInputOutput

Fornisce un canale di comunicazione per instradare output e input dello script in esecuzione, ma agisce tramite eventi.

- ⚡ **OnReceiveData**: viene generato quando l'interprete Python è in attesa di dati in input
- ⚡ **OnSendData**: viene generato quando l'interprete Python invia dati all'output



NOTA: è necessario una applicazione Console! `{ $APPTYPE CONSOLE }`



# Proviamo!





# TPythonModule

Consente di creare moduli all'interno del runtime di Python, definendone il nome e le funzioni che ne fanno parte.

- ⚙ AddMethod()
- ⚙ AddMethodWithKW()
- ⚙ SetVar()





# TPythonDelphiVar: panoramica

Crea una variabile Python all'interno del runtime e consente di manipolarla tramite Delphi.

- Module
- VarName
  
- ValueAsString
  
- OnChange





# Proviamo!





# TPyDelphiWrapper

Consente di creare un “wrapper” attorno a record, oggetti, classi e così via, allo scopo di renderli accessibili all’ambiente Python.

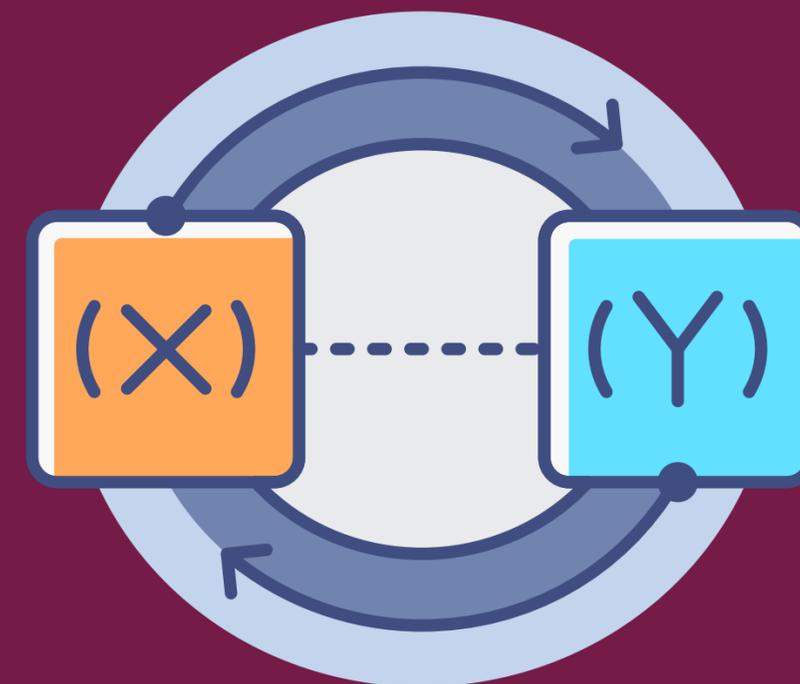
- Wrap()
- WrapClass()
- WrapRecord()





# VarPyth

- Accesso ad alto livello agli oggetti Python da Delphi
- Creazione di oggetti tipici di Python (liste, tuple, ecc.)
- Uso di custom variant
- Non occorre usare l'API di basso livello di Python o gestire i conteggi dei riferimenti agli oggetti
- Impatto lieve sulle performance





# Proviamo!





**Da scripting ad  
applicazioni full**

**4**



# Utilizzo di librerie Python

---

## Data analysis

- numpy
- scipy
- pandas

## Data visualization

- matplotlib: plotting di grafici
- seaborn: visualizzazione dati statistici

## Machine Learning

- TensorFlow
- PyTorch
- Scikit-learn

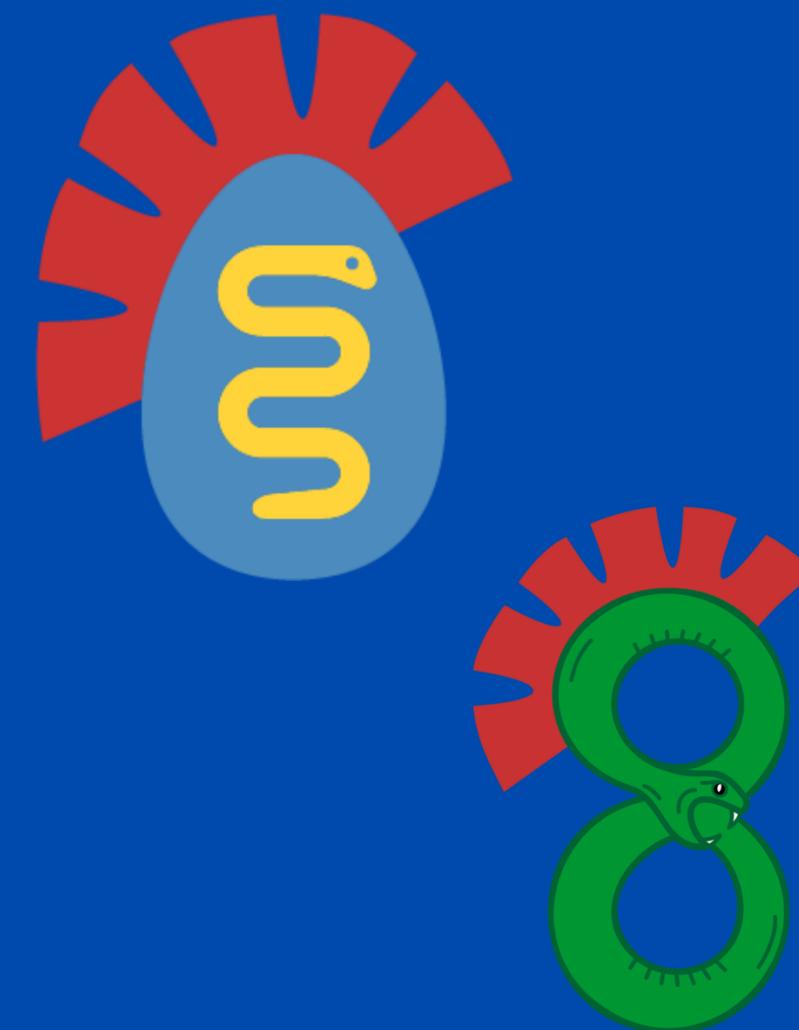




# Embarcadero Add-Ons



- **PythonEnvironments**
  - Componenti designtime per il supporto ai “virtual environment” di Python
- **Python Packages for Delphi**
  - Collezione di package “wrappati” per un uso immediato in Delphi
- **P4D-Data-Sciences**
  - Wrapper leggeri per il Data Science in Delphi



👉 <https://github.com/Embarcadero/PythonPackages4Delphi>



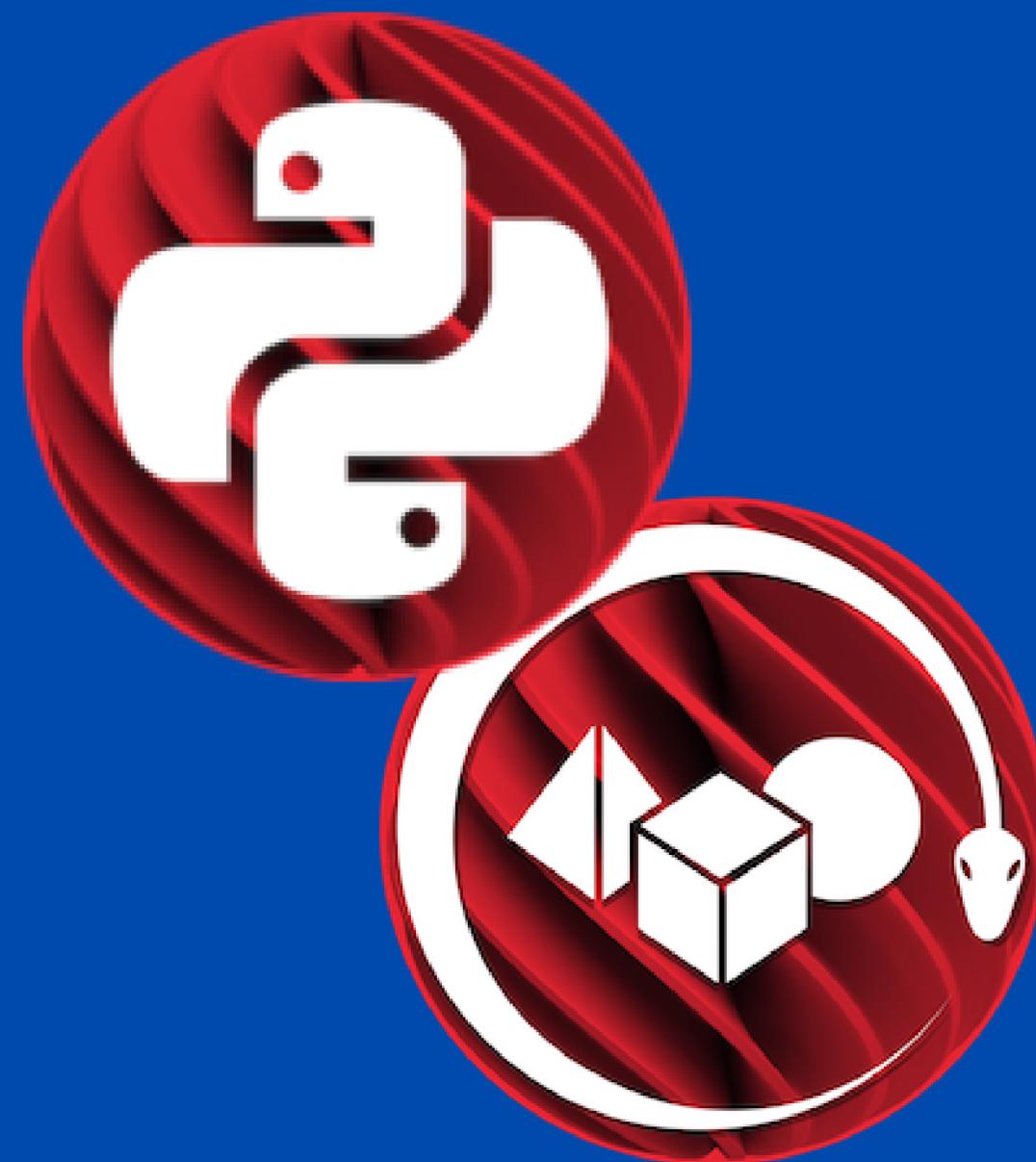
# Delphi 4 Python VCL/FMX

- **Delphi VCL/FMX for Python**
  - Sono librerie native e compilate con P4D per consentire agli sviluppatori Python di sfruttare VCL e FMX per creare le GUI delle proprie applicazioni
- **Delphi4PythonExporter**
  - Disegna la GUI per i programmi Python con Delphi usando VCL e FMX poi esportali in Python

👉 <https://github.com/Embarcadero/DelphiVCL4Python>

👉 <https://github.com/Embarcadero/DelphiFMX4Python>

👉 <https://github.com/Embarcadero/PythonPackages4Delphi>





**Proviamo!**





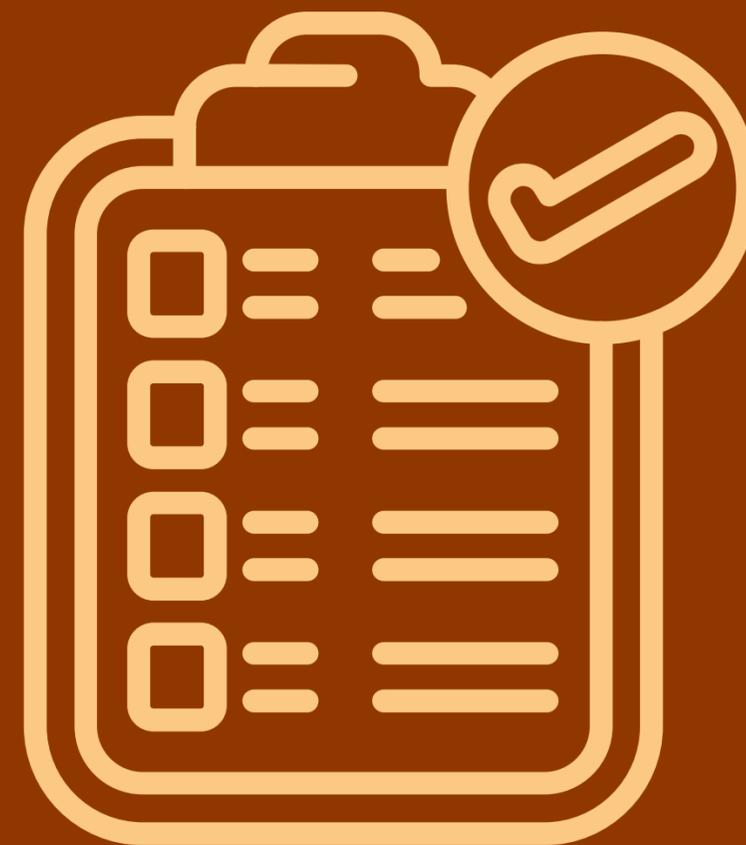
# Wrap-up!

# 5



# Conclusioni

- Grazie all'integrazione con il linguaggio Python, enormemente agevolato dalla libreria Python4Delphi, è possibile ottenere il meglio dei "due mondi"
- Qualunque funzionalità o libreria mancante in Delphi può essere facilmente implementata grazie ai package già disponibili per l'ecosistema Python
- Il supporto GUI per Python è molto frammentato: grazie all'integrazione con Delphi, è possibile sviluppare GUI funzionali per applicazioni Python
- L'integrazione con Python è perfetta per tutte le esigenze di sviluppo Delphi che richiedono scripting dinamico.





# Risorse e approfondimenti /1

---

**Sito ufficiale di Python**

👉 <https://www.python.org/>

**Documentazione ufficiale di Python**

👉 <https://docs.python.org/3/library/index.html>

**Libro “Python Crash Course (2nd Edition)”**

👉 [https://ehmatthes.github.io/pcc\\_2e/regular\\_index/](https://ehmatthes.github.io/pcc_2e/regular_index/)

**Libro “Automate The Boring Stuff with Python”**

👉 <https://automatetheboringstuff.com/>





# Risorse e approfondimenti /2

---

## Libreria Python4Delphi

👉 <https://github.com/pyscripter/python4delphi>

## PyScripter: IDE gratuito e open-source per Python

👉 <https://github.com/pyscripter/pyscripter>

## Webinar e risorse Embarcadero su Delphi+Python

👉 <https://blogs.embarcadero.com/python-for-delphi-developers-webinar/>

## Python GUI.org by Embarcadero

👉 <https://pythongui.org/easily-build-powerful-python-gui-modules-in-delphi-with-this-windows-sample-app/>





# Domande?





**THANK YOU**