Funzioni e Moduli in Python

Le funzioni e i moduli rappresentano i pilastri fondamentali della programmazione Python moderna. Questi strumenti permettono di organizzare il codice in modo logico, riutilizzabile e facilmente manutenibile. Durante questa lezione, esploreremo come creare funzioni efficaci, gestire parametri e valori di ritorno, e utilizzare i moduli per strutturare progetti complessi. Impareremo anche le migliori pratiche per scrivere codice pulito e professionale.

Cosa sono le Funzioni

Definizione

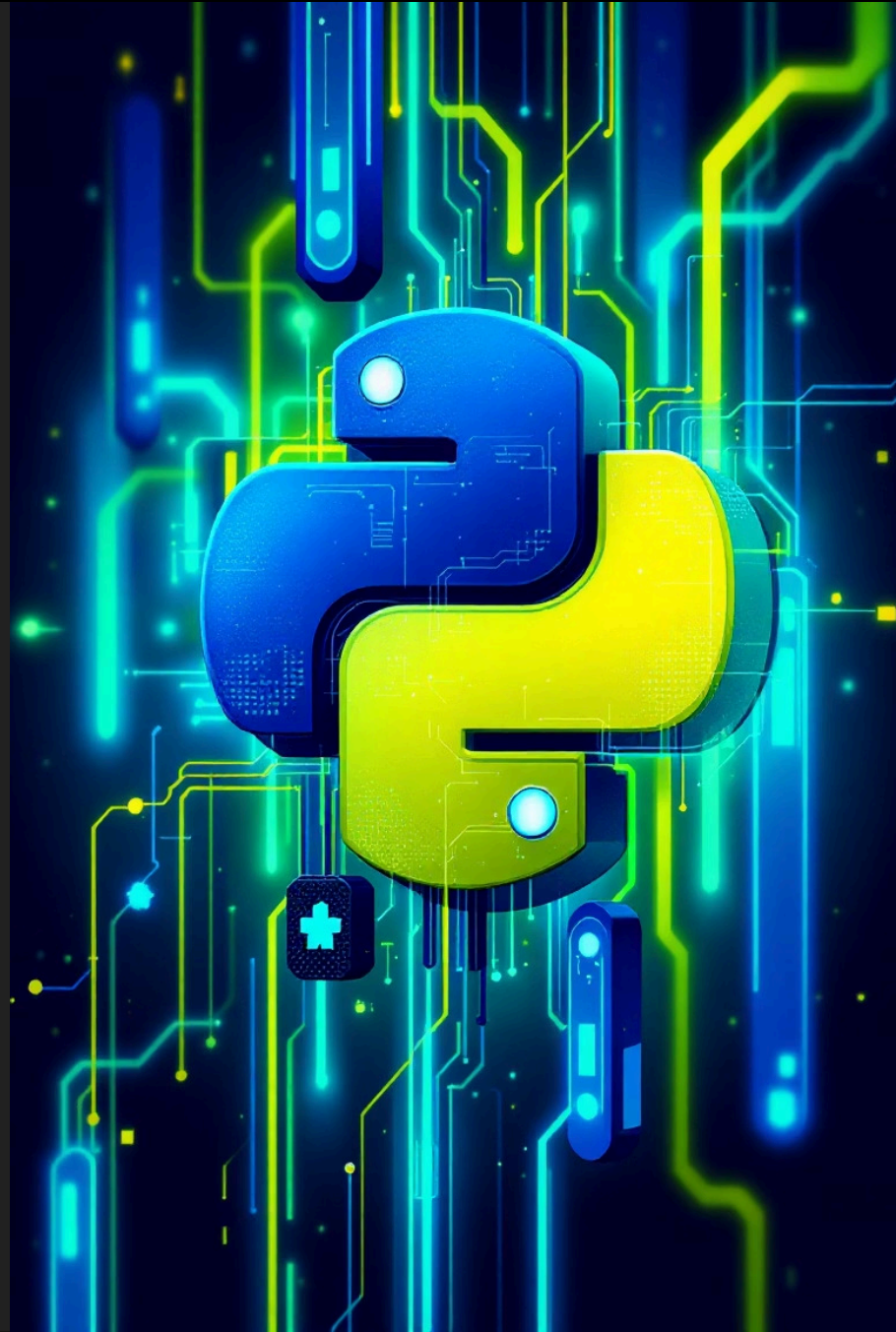
Una funzione è un blocco di codice riutilizzabile che esegue un compito specifico. È come una macchina che riceve input, li elabora e produce un output.

Vantaggi

Le funzioni riducono la duplicazione del codice, migliorano la leggibilità e facilitano il debugging. Rendono il programma modulare e organizzato.

Analogia

Pensate a una funzione come a una ricetta di cucina: ha ingredienti (parametri), istruzioni (corpo della funzione) e un piatto finale (valore di ritorno).



Definizione di Funzione con def

1

Parola chiave def

Inizia sempre con la keyword 'def' seguita dal nome della funzione

2

Nome funzione

Deve essere descrittivo e seguire le convenzioni Python (snake_case)

3

Parentesi e parametri

Le parentesi contengono i parametri, anche se vuote sono obbligatorie

4

Due punti e corpo

I due punti indicano l'inizio del blocco, indentato di 4 spazi

La sintassi base è: def nome_funzione(parametri): seguito dal corpo della funzione indentato. Ricordate che l'indentazione in Python non è solo estetica, ma è parte della sintassi del linguaggio.

Parametri e Argomenti

Parametri

I parametri sono le variabili definite nella dichiarazione della funzione. Agiscono come placeholder per i valori che la funzione riceverà quando viene chiamata.

- Parametri posizionali: `def saluta(nome):`
- Parametri con valore predefinito: `def saluta(nome="Utente"):`
- Parametri keyword: `def crea_profilo(nome, età, città):`

```
def calcola(a, b=0, c=1):
    return a + b * c
```

Argomenti

Gli argomenti sono i valori effettivi passati alla funzione durante la chiamata. Possono essere valori letterali, variabili o espressioni.

- Argomenti posizionali: `saluta("Marco")`
- Argomenti keyword: `crea_profilo(nome="Laura", età=29, città="Milano")`
- Argomenti variabili: `def somma_tutto(*numeri, **opzioni):`

```
risultato = calcola(5)      # a=5, b=0, c=1
risultato = calcola(5, 3)   # a=5, b=3, c=1
risultato = calcola(5, c=3) # a=5, b=0, c=3
```

Valori di Ritorno con return

1 Return esplicito

La parola chiave return restituisce un valore e termina immediatamente l'esecuzione della funzione

2 Return implicito

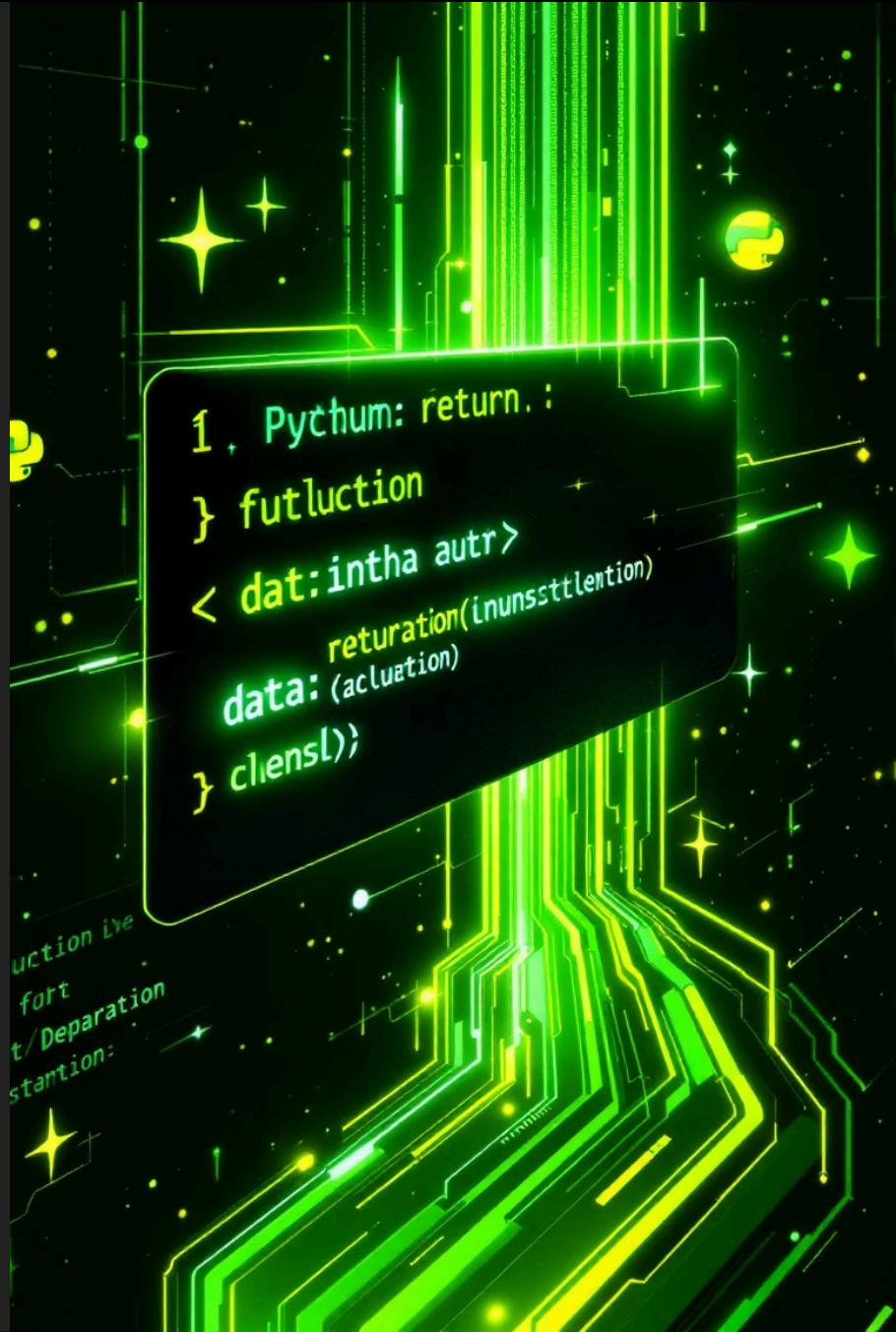
Se non specificate return, Python restituisce automaticamente None al termine della funzione

3 Valori multipli

Potete restituire più valori separandoli con virgole, Python li impacchetta in una tupla

4 Return condizionale

È possibile avere più istruzioni return basate su condizioni diverse all'interno della stessa funzione

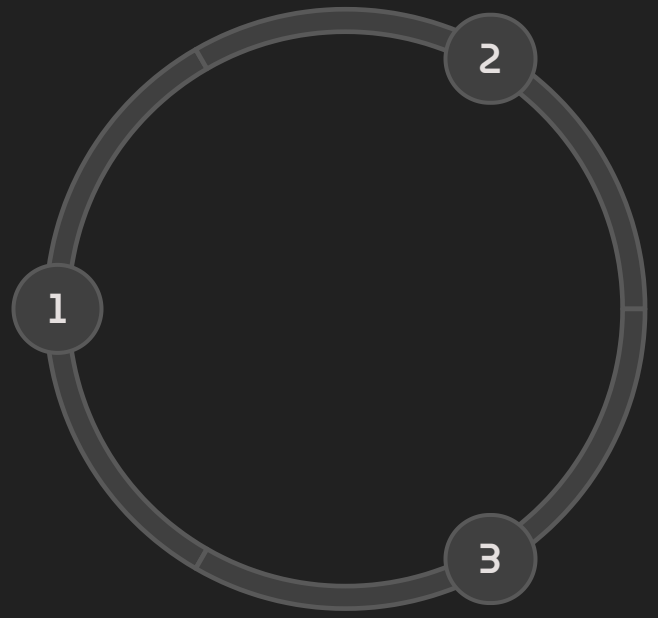


Scope: Variabili Locali e Globali

Scope Locale

Variabili create dentro una funzione

- Visibili solo nella funzione
- Distrutte al termine della funzione



Scope Globale

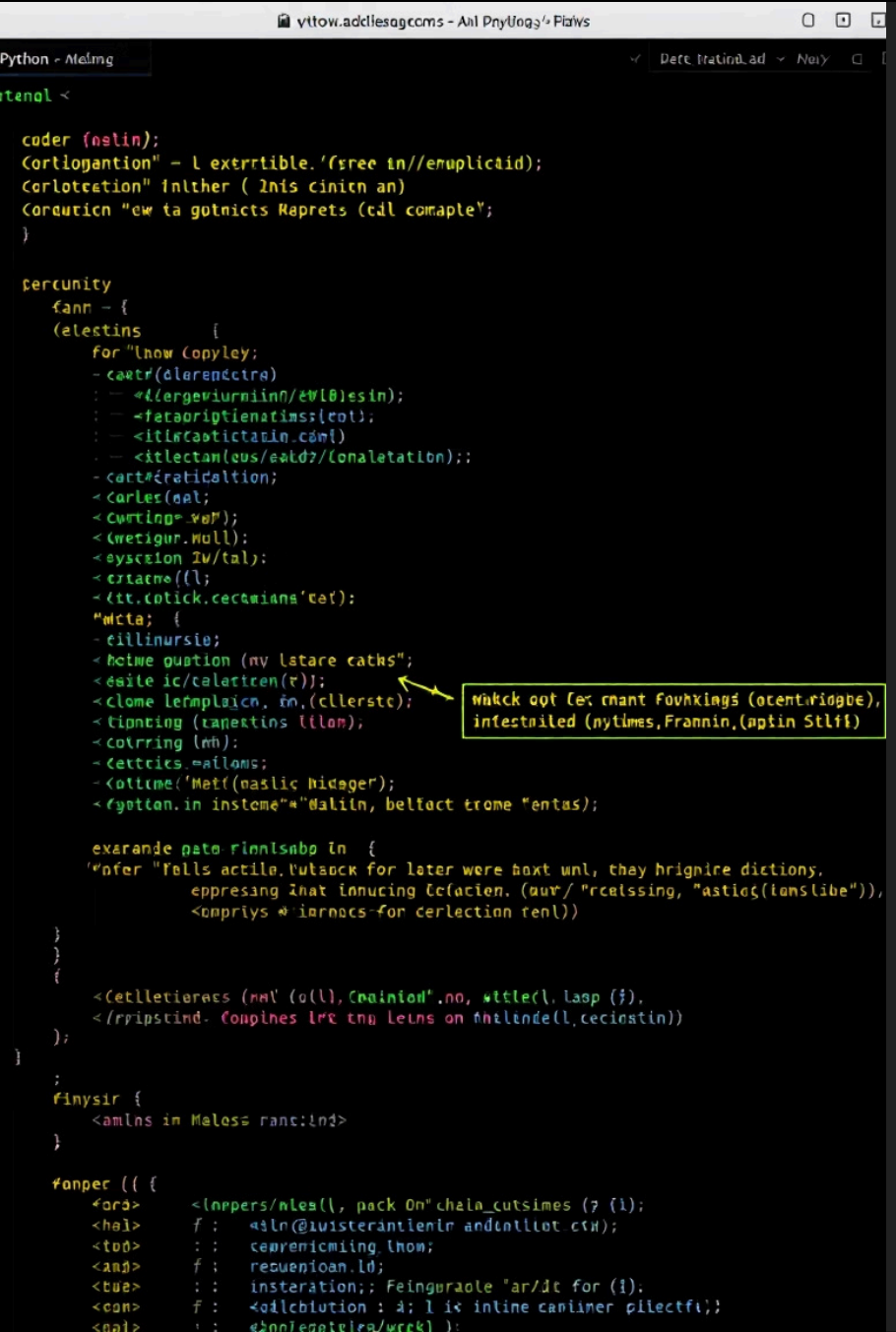
Variabili create nel programma principale

- Accessibili ovunque nel modulo
- Persistono per tutta l'esecuzione

Regola LEGB

Local, Enclosing, Global, Built-in

- Python cerca le variabili in quest'ordine
- Importante per risolvere conflitti di nomi



Documentazione Funzioni (Docstring)

Posizionamento

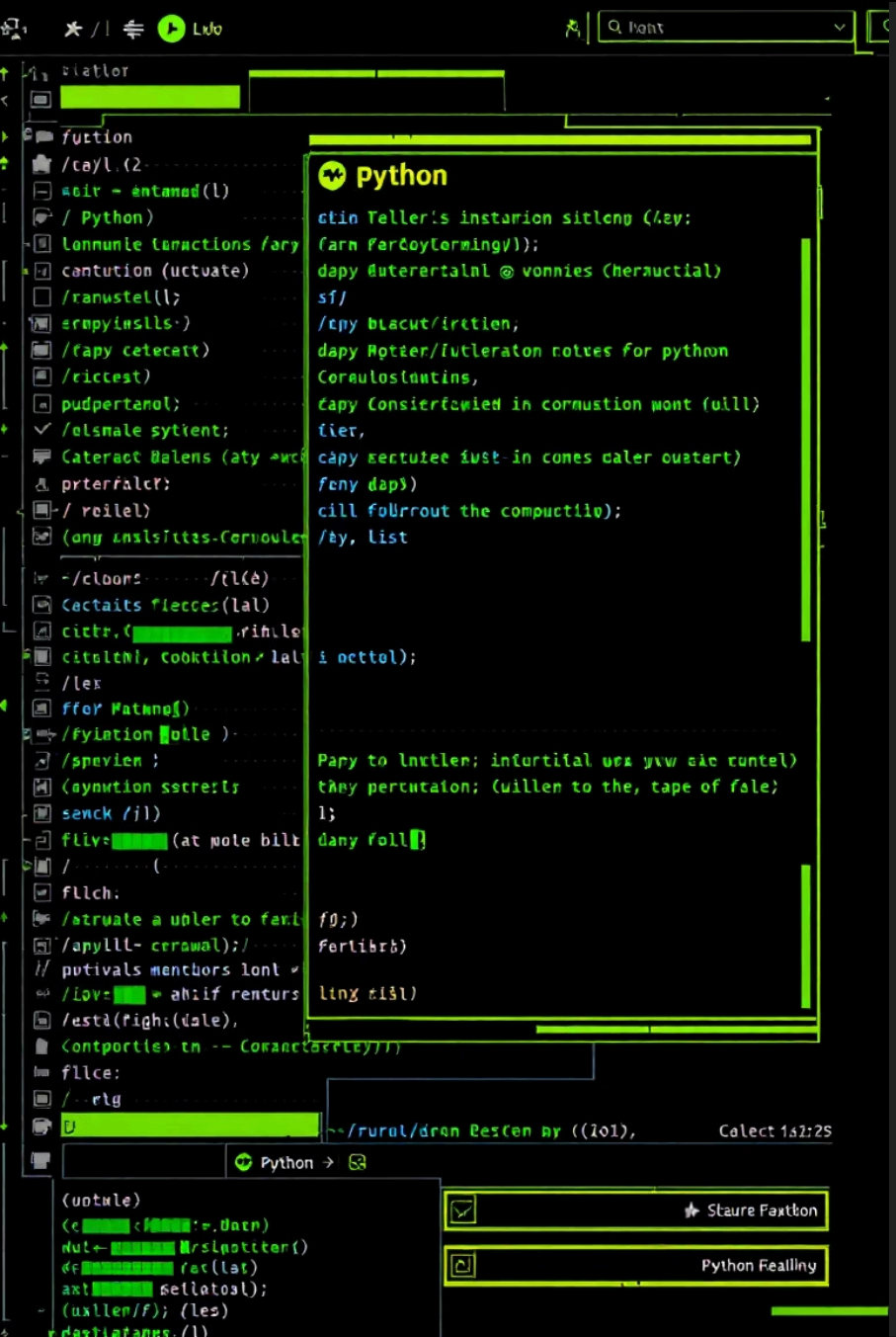
Il docstring va inserito come prima riga del corpo della funzione, racchiuso tra triple virgolette. Deve essere immediatamente dopo la definizione della funzione.

Contenuto Standard

Descrivete cosa fa la funzione, i parametri accettati con i loro tipi, e cosa restituisce. Include esempi di utilizzo quando appropriato.

Accessibilità

I docstring diventano accessibili tramite l'attributo __doc__ della funzione e vengono utilizzati da help() e strumenti di documentazione automatica.



Buone Pratiche di Naming e Modularità

Nomi Descrittivi

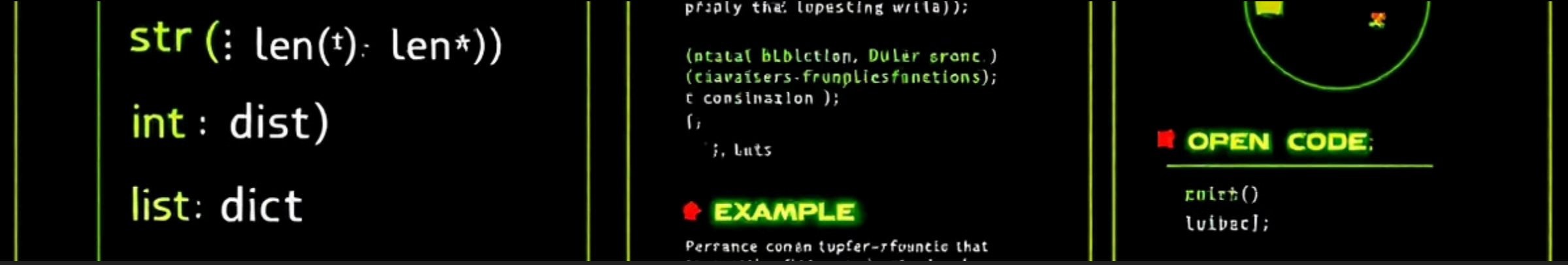
Utilizzate nomi che descrivono chiaramente il compito della funzione. Preferite nomi lunghi ma chiari a abbreviazioni cryptiche. Seguite la convenzione snake_case per i nomi delle funzioni.

Singola Responsabilità

Ogni funzione dovrebbe fare una sola cosa e farla bene. Se una funzione diventa troppo complessa, dividetela in funzioni più piccole e specifiche.

Lunghezza Appropriata

Mantenete le funzioni concise, generalmente sotto le 20-30 righe. Funzioni più lunghe sono difficili da testare, comprendere e mantenere nel tempo.



Funzioni Built-in di Python

Funzioni di Base

print(), input(), len(), type(), str(), int(), float() - le funzioni essenziali per iniziare

1

Funzioni Matematiche

2

abs(), min(), max(), sum(), round() - operazioni matematiche comuni senza import aggiuntivi

Funzioni di Sequenza

3

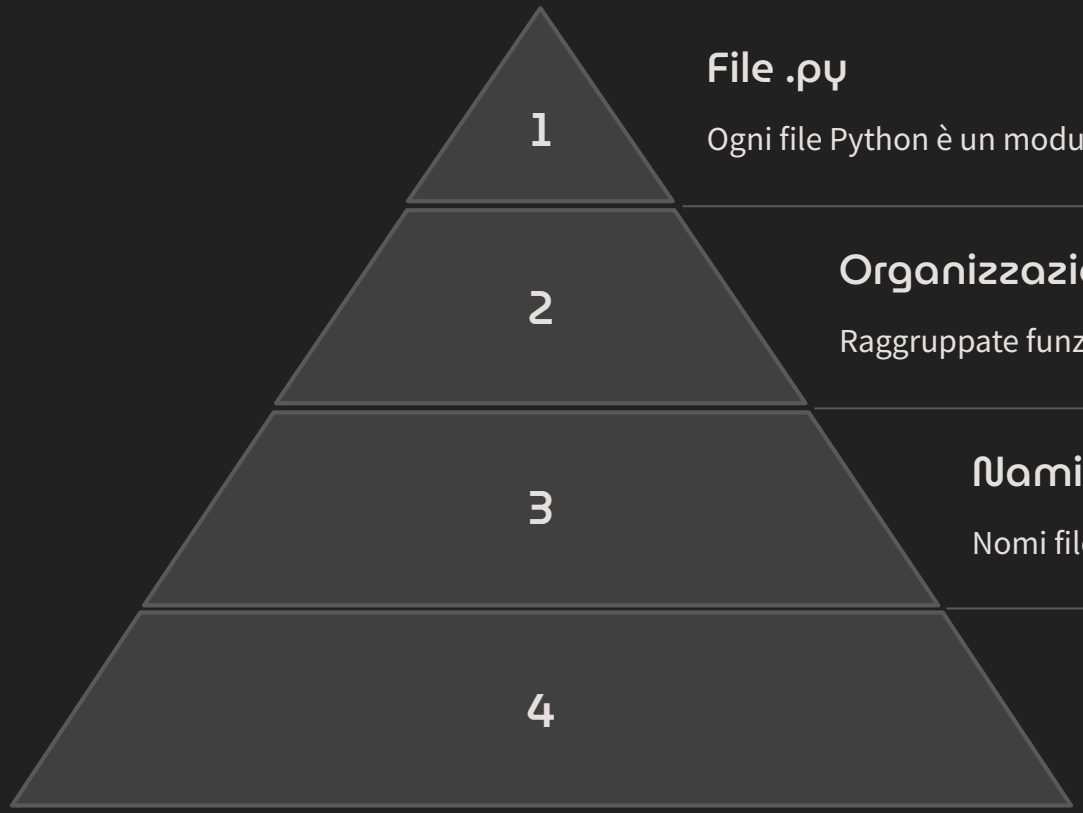
range(), enumerate(), zip(), sorted(), reversed() - per lavorare con liste, tuple e stringhe

Funzioni Avanzate

4

map(), filter(), any(), all() - programmazione funzionale e operazioni su collezioni

Creazione di Moduli Personalizzati



File .py

Ogni file Python è un modulo

Organizzazione

Raggruppate funzioni correlate nello stesso modulo

Naming

Nomi file descrittivi senza spazi o caratteri speciali

Struttura

Documentazione, import, costanti, funzioni, codice principale

Un modulo personalizzato è semplicemente un file Python che contiene definizioni di funzioni, classi e variabili. La creazione di moduli personalizzati vi permette di riutilizzare il codice tra progetti diversi e di organizzare meglio il vostro lavoro in componenti logici e gestibili.

Import di Moduli Standard

Modulo Math

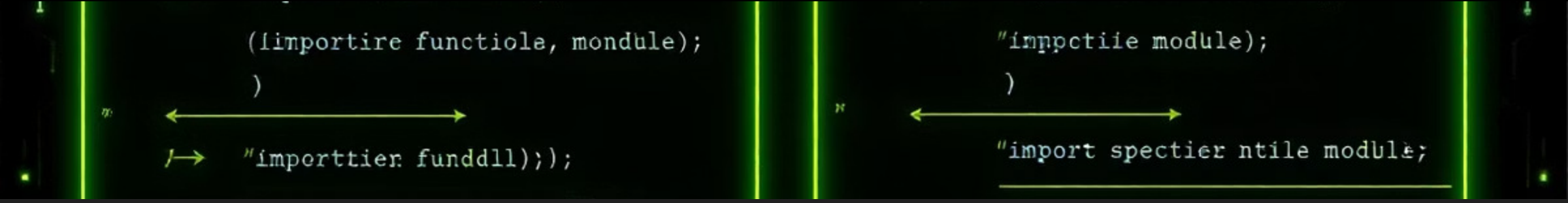
Il modulo math fornisce funzioni matematiche avanzate come sin(), cos(), sqrt(), pi, e. È essenziale per calcoli scientifici e ingegneristici.

- Funzioni trigonometriche
- Logaritmi e esponenziali
- Costanti matematiche

Modulo Random

Il modulo random permette di generare numeri casuali e fare scelte random. Fondamentale per simulazioni, giochi e testing.

- random() per numeri 0-1
- randint() per interi
- choice() per elementi da liste



Differenza tra import e from-import

1. **import modulo**

   Importa l'intero modulo, accesso tramite modulo.funzione(). Mantiene il namespace pulito ma richiede prefisso.

2. **from modulo import funzione**

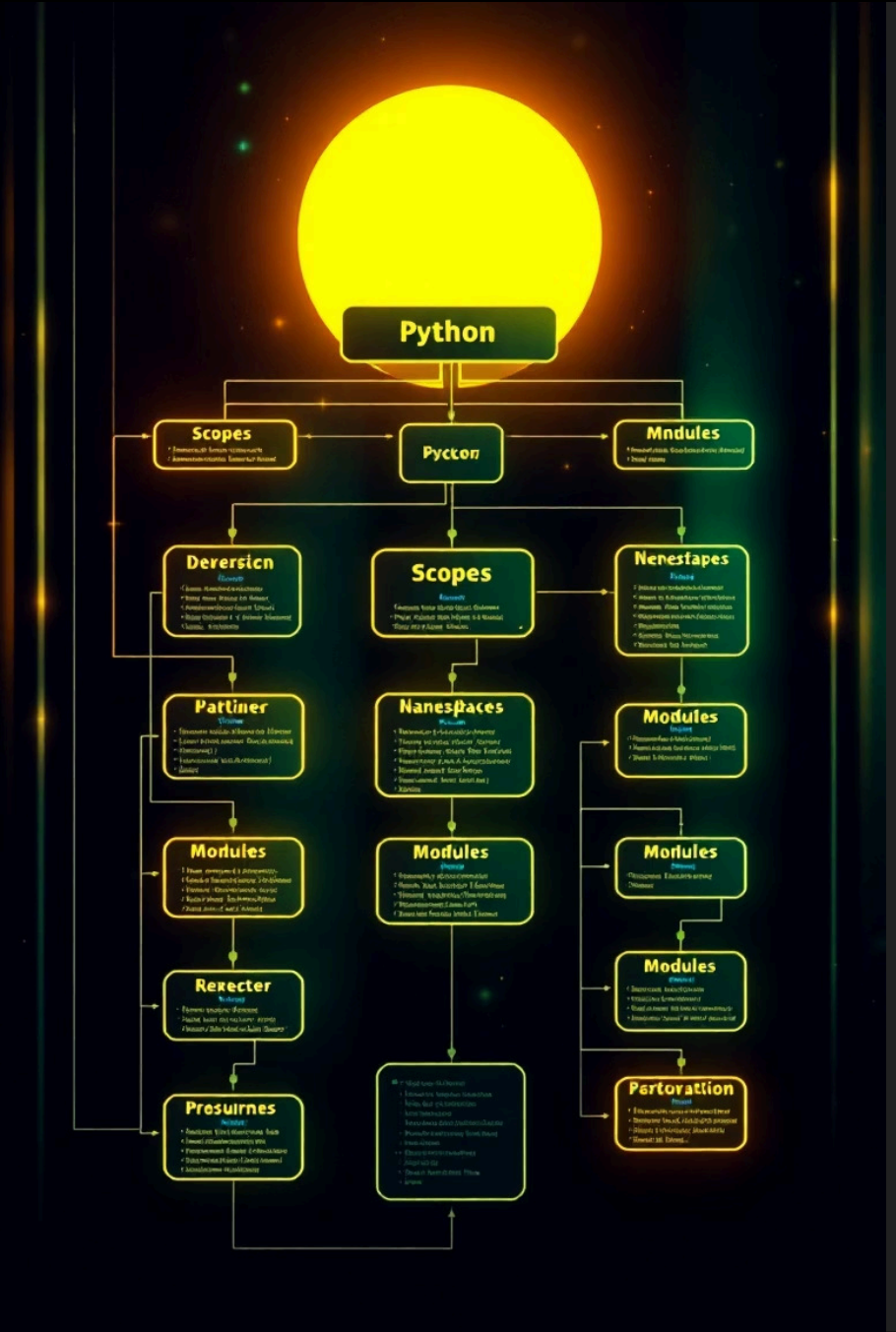
   Importa specifiche funzioni nel namespace corrente. Accesso diretto senza prefisso, ma rischio di conflitti di nomi.

3. **from modulo import ***

   Importa tutto dal modulo. Sconsigliato perché inquina il namespace e rende difficile tracciare l'origine delle funzioni.

4. **import modulo as alias**

   Crea un alias per il modulo. Utile per moduli con nomi lunghi o per evitare conflitti mantenendo il namespace pulito.



Gestione del Namespace

Namespace Locale

Contiene nomi definiti all'interno di una funzione. Questi nomi sono accessibili solo durante l'esecuzione della funzione e vengono eliminati al termine.

Namespace Globale

Contiene nomi definiti a livello di modulo. Include funzioni, classi e variabili definite nel file principale, accessibili da qualsiasi punto del modulo.

Namespace Built-in

Contiene nomi predefiniti di Python come print, len, type. Sempre accessibile e caricato automaticamente all'avvio dell'interprete Python.

Risoluzione Nomi

Python cerca i nomi seguendo l'ordine LEGB: Local, Enclosing, Global, Built-in. Questa gerarchia determina quale variabile viene utilizzata in caso di conflitti.

Condizione if __name__ == "__main__"

Esecuzione Diretta

Quando un file Python viene eseguito direttamente, __name__ è impostato a "__main__"

Import come Modulo

Quando il file viene importato, __name__ contiene il nome del modulo invece di "__main__"

Best Practice

Permette di creare moduli che possono essere sia importati che eseguiti autonomamente

Codice Condizionale

Il blocco if __name__ == "__main__": viene eseguito solo se il file è lanciato direttamente

Questa condizione è fondamentale per scrivere codice modulare. Vi permette di includere codice di test o esempi che si eseguono solo quando il file viene lanciato direttamente, ma non quando viene importato da altri moduli.

Uso di help() per Moduli e Funzioni

3

Modi di Utilizzo

help(), help(oggetto), help('stringa') per diversi tipi di documentazione

100+

Moduli Disponibili

Centinaia di moduli standard documentati attraverso help()

24/7

Disponibilità

Documentazione sempre accessibile senza connessione internet

La funzione help() è uno strumento preziosissimo per esplorare Python. Fornisce documentazione dettagliata per qualsiasi oggetto: funzioni, moduli, classi e metodi. Potete usare help() senza argomenti per entrare in modalità interattiva, help(nome_funzione) per documentazione specifica, o help('keywords') per vedere le parole chiave di Python. È particolarmente utile quando lavorate con moduli nuovi o quando volete ricordare la sintassi di funzioni built-in. La documentazione mostrata include descrizioni, parametri, valori di ritorno ed esempi di utilizzo quando disponibili.