Tipi Primitivi e Operatori in Python

Benvenuti alla lezione fondamentale sui tipi primitivi e operatori in Python! Oggi esploreremo i mattoni fondamentali della programmazione: i diversi tipi di dati che Python può gestire e le operazioni che possiamo eseguire su di essi. Questa conoscenza è essenziale per scrivere programmi efficaci e comprendere come il computer elabora le informazioni.

Durante questa lezione imparerete a distinguere tra numeri interi, decimali, testo e valori booleani, scoprendo come Python li gestisce internamente. Vedremo anche come combinare questi elementi usando operatori matematici, logici e di confronto per creare programmi più complessi e interessanti.

Obiettivi della Lezione

1 **Comprendere i Tipi Primitivi**

Imparare a riconoscere e utilizzare int, float, str e bool in Python

2 **Padroneggiare gli Operatori**

Utilizzare operatori aritmetici, logici e di confronto correttamente

3 **Gestire le Conversioni**

Convertire tra diversi tipi di dati ed evitare errori comuni

4 **Applicare le Conoscenze**

Scrivere codice pulito usando buone pratiche di denominazione e formattazione

I Quattro Tipi Primitivi Fondamentali

int (Numeri Interi)

Rappresentano numeri senza decimali come 42, -17, 0. Perfetti per conteggi, indici e calcoli che richiedono precisione assoluta. Python gestisce automaticamente numeri interi di qualsiasi dimensione.

float (Numeri Decimali)

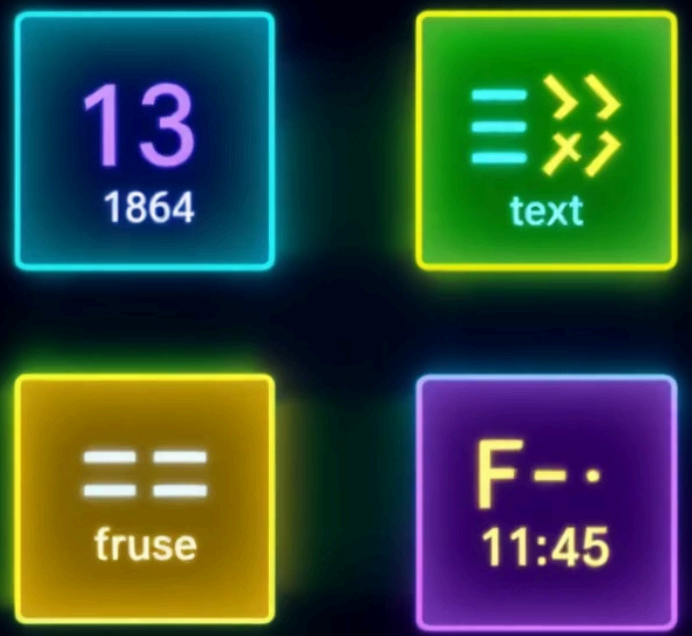
Numeri con virgola come 3.14, -2.5, 0.001. Utilizzati per misurazioni, calcoli scientifici e quando serve precisione decimale. Attenzione agli errori di arrotondamento!

str (Stringhe di Testo)

Sequenze di caratteri racchiuse tra virgolette: "Ciao", 'Python', "123". Usate per nomi, messaggi, input utente. Immutabili una volta create.

bool (Valori Booleani)

Solo True o False (maiuscole obbligatorie). Fondamentali per decisioni, condizioni e logica del programma. Risultato di confronti e operazioni logiche.



Differenze Cruciali tra i Tipi

Precisione e Memoria

Gli int offrono precisione assoluta e occupano memoria variabile. I float hanno precisione limitata (circa 15-17 cifre significative) ma permettono calcoli decimali. Le stringhe memorizzano caratteri Unicode, mentre i bool occupano lo spazio minimo.

Esempio: 1/3 come float diventa 0.3333333333333333, perdendo precisione rispetto alla frazione matematica esatta.

Operazioni Supportate

Ogni tipo supporta operazioni specifiche. I numeri permettono calcoli matematici, le stringhe si possono concatenare e ripetere, i bool si combinano con operatori logici.

Attenzione: "5" + "3" produce "53" (concatenazione), mentre 5 + 3 produce 8 (addizione). La comprensione di queste differenze previene errori comuni.

Conversioni tra Tipi (Casting)

1

Verso Numeri

int("42") converte stringa in intero. float("3.14") crea un decimale. Attenzione: int("3.14") genera errore!

2

Verso Stringhe

str(42) produce "42". str(3.14) diventa "3.14". str(True) risulta in "True". Sempre sicuro, non genera mai errori.

3

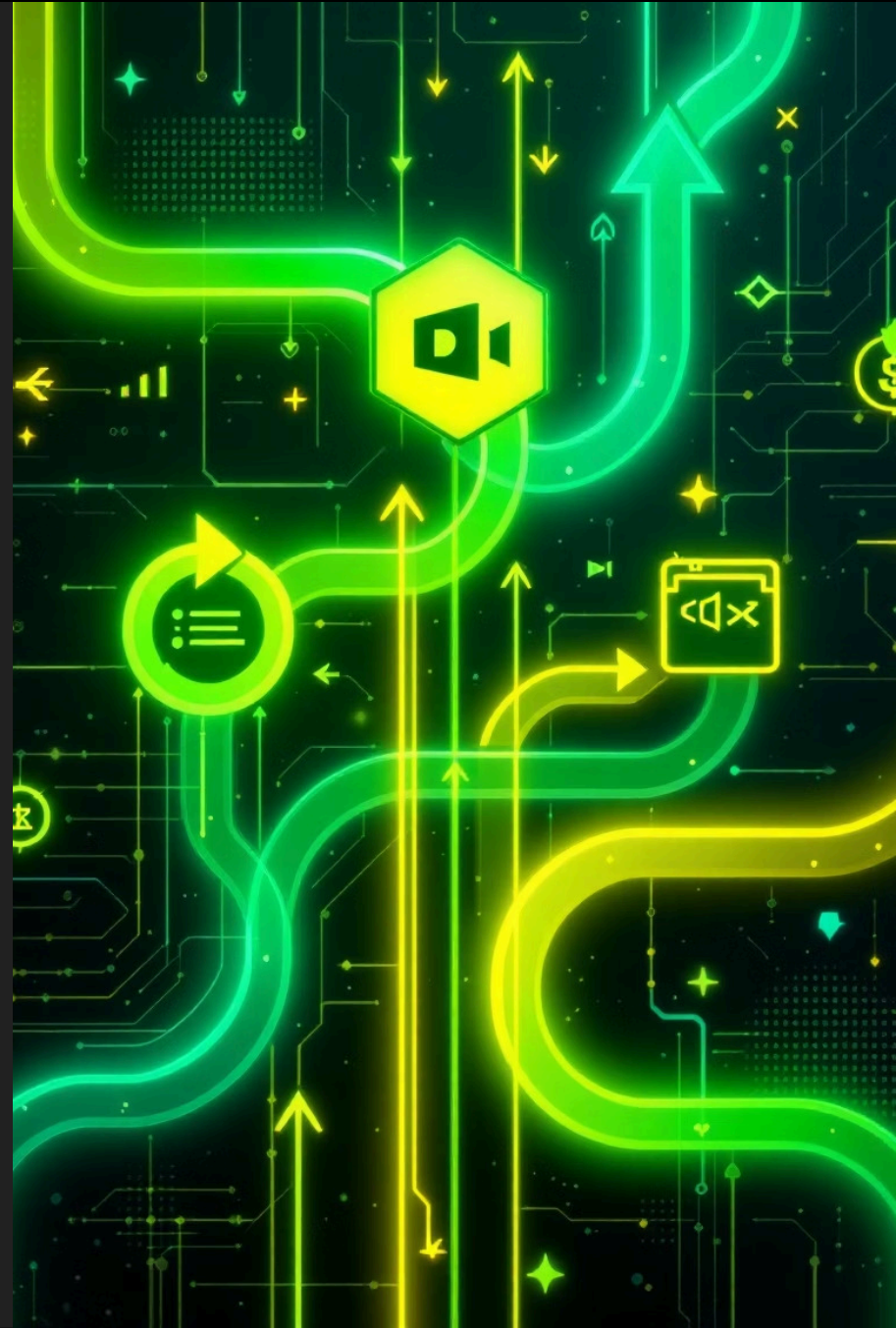
Verso Booleani

bool(0) è False, bool(1) è True. bool("") è False, bool("testo") è True. Regola: valori "vuoti" sono False.

4

Conversioni Automatiche

Python converte automaticamente int a float in operazioni miste: 5 + 3.2 = 8.2 (float). Sempre verso il tipo più "generale".



Operatori Aritmetici Essenziali

Addizione (+)

Somma numeri o concatena stringhe. 5 + 3 = 8, "Ciao" + " mondo" = "Ciao mondo"

Sottrazione (-)

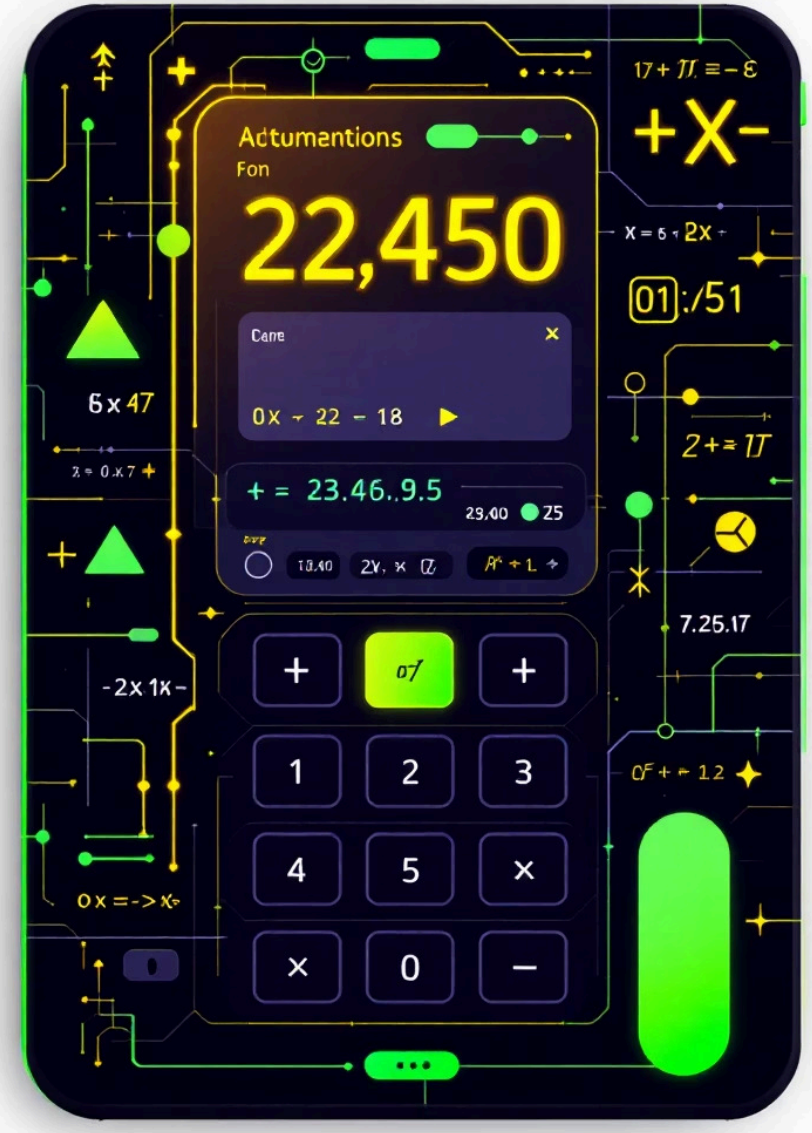
Solo per numeri. 10 - 3 = 7. Può rendere negativi i risultati.

Moltiplicazione (*)

Numeri: 4 * 5 = 20. Stringhe: "Ha" * 3 = "HaHaHa". Utile per ripetizioni!

Divisione (/)

Sempre produce float. 10 / 3 = 3.3333... Attenzione alla divisione per zero!



Operatori Aritmetici Avanzati

1 — **Modulo (%)**

Resto della divisione. 10 % 3 = 1. Utile per verificare numeri pari/dispari: n % 2 == 0 significa pari.

2 — **Divisione Intera (//)**

Parte intera della divisione. 10 // 3 = 3. Elimina la parte decimale, diverso da int(10/3).

3 — **Potenza (**)**

Elevamento a potenza. 2 ** 3 = 8, 9 ** 0.5 = 3.0. Supporta anche radici con esponenti decimali.

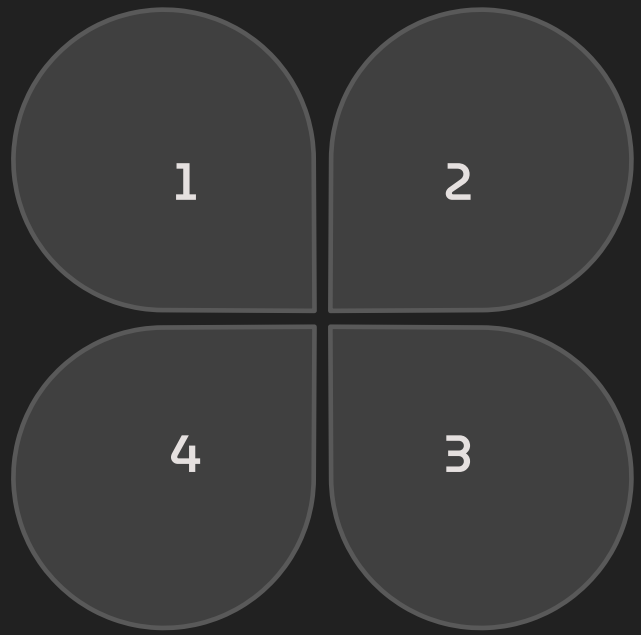
Operatori di Confronto

Uguaglianza (==)

Verifica se due valori sono identici. 5 == 5 è True, "a" == "b" è False. Attenzione: = è assegnazione, == è confronto!

Minore/Maggiore Uguale (<=, >=)

Include l'uguaglianza. 5 <= 5 è True. Molto comuni nei cicli e nelle condizioni.



Disuguaglianza (!=)

Opposto di ==. 5 != 3 è True. Molto utile per verificare condizioni negative.

Minore/Maggiore (<, >)

Confronti numerici e lessicografici. 5 < 10 è True, "apple" < "banana" è True (ordine alfabetico).

Operatori Logici Fondamentali

AND Logico

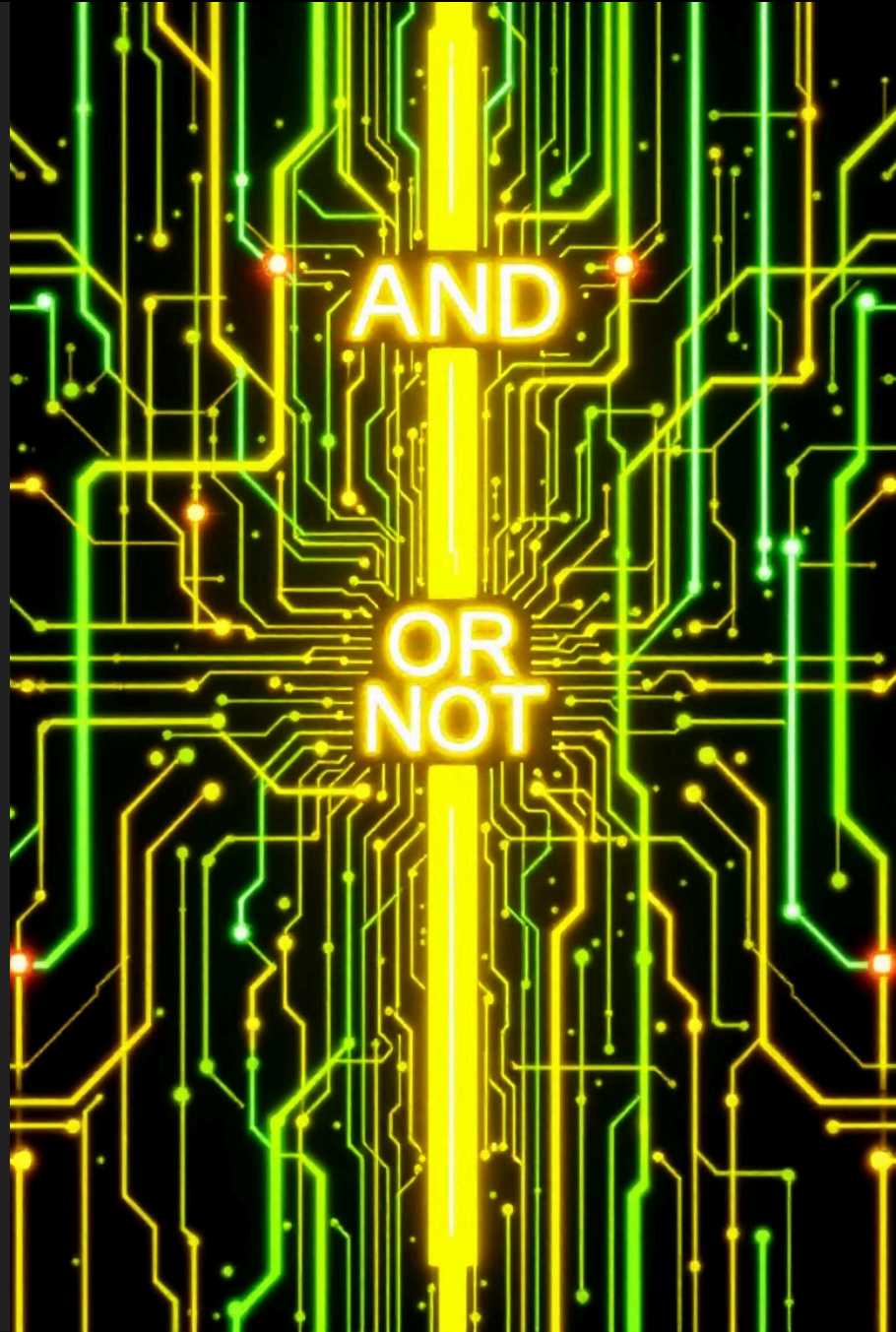
True solo se entrambi gli operandi sono True. True and True = True, tutto il resto è False. Utile per verificare multiple condizioni simultaneamente.

OR Logico

True se almeno uno degli operandi è True. False or True = True. Solo False or False = False. Perfetto per condizioni alternative.

NOT Logico

Inverte il valore booleano. not True = False, not False = True. Essenziale per negare condizioni e creare logica inversa.



Precedenza degli Operatori

Parentesi ()

Priorità assoluta

Potenza **

Operazioni esponenziali

***, /, //, %**

Moltiplicazione e divisioni

+, -

Addizione e sottrazione

Confronti e Logici

==, !=, <, >, and, or, not

Assegnazione e Aggiornamento Variabili

Assegnazione Semplice

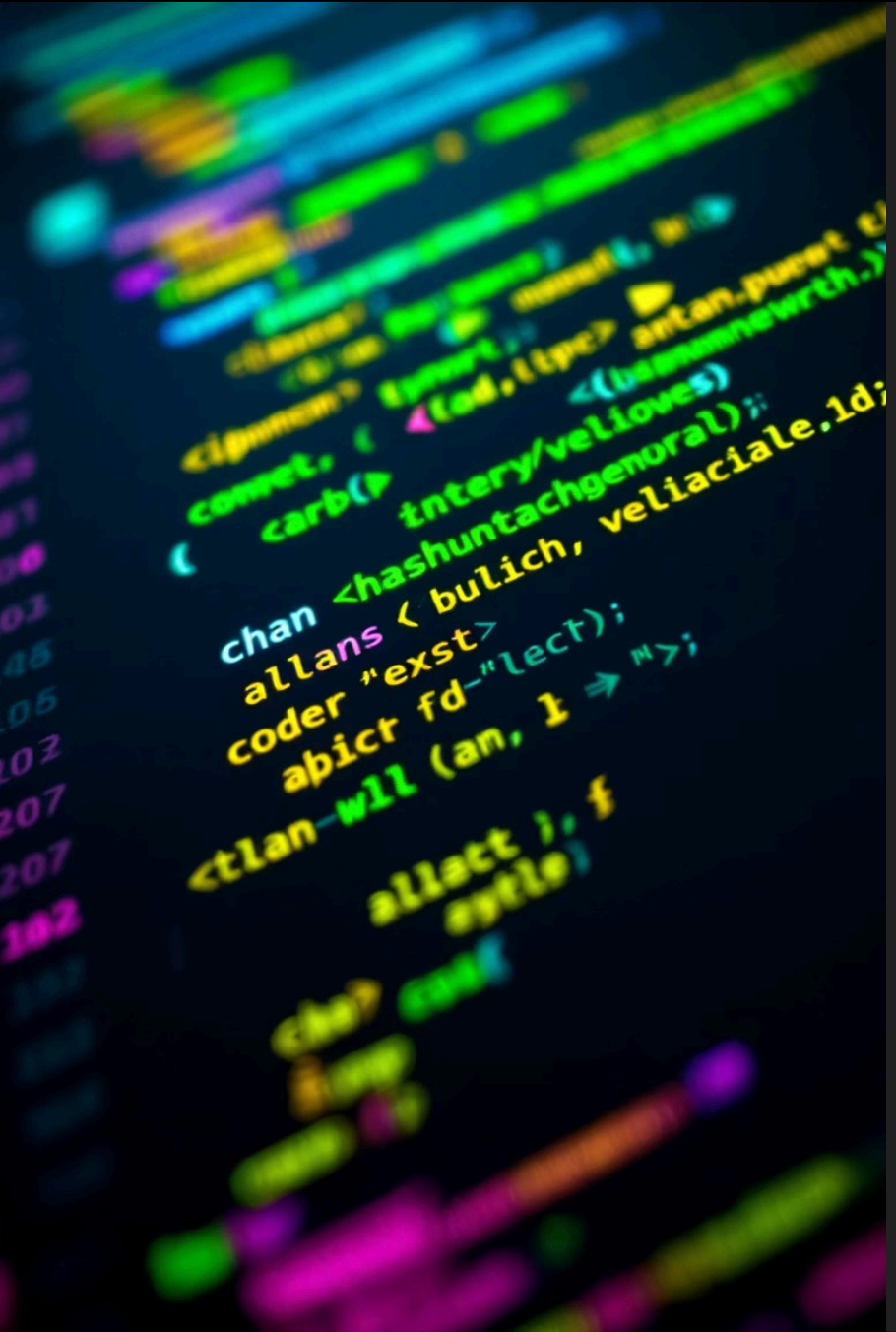
x = 5 crea una variabile chiamata x e le assegna il valore 5. Il nome della variabile diventa un'etichetta che punta a quel valore in memoria.

Python è tipizzato dinamicamente: la stessa variabile può contenere tipi diversi in momenti diversi. x = 5, poi x = "ciao" è perfettamente valido.

Operatori di Aggiornamento

Gli operatori combinati semplificano l'aggiornamento: x += 3 equivale a x = x + 3. Disponibili anche -=, *=, /=, //=, %=, **=.

Questi operatori rendono il codice più leggibile e esprimono chiaramente l'intenzione di modificare una variabile esistente piuttosto che assegnarne una nuova.



Costanti e Nomi Significativi

Convenzioni per Costanti

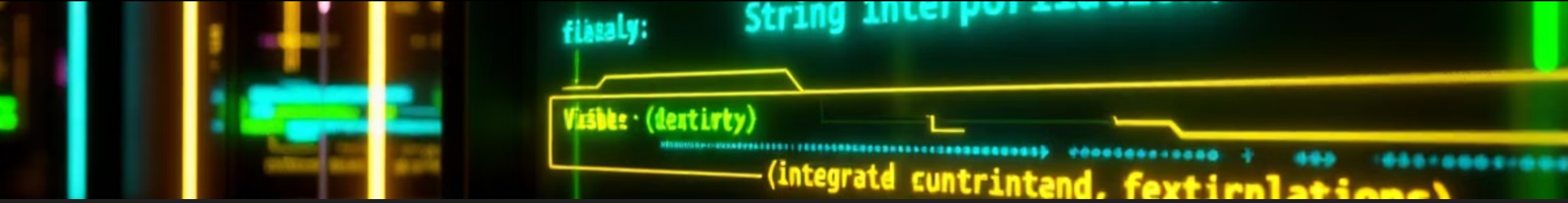
In Python, le costanti si scrivono in MAIUSCOLO: PI = 3.14159, MAX_USERS = 100. Anche se Python non impedisce la modifica, la convenzione indica che non dovrebbero cambiare.

Nomi Descrittivi

Usate nomi che descrivono il contenuto: eta_studente invece di x, prezzo_totale invece di p. Il codice deve essere auto-documentante e comprensibile anche dopo mesi.

Convenzioni di Naming

Usate snake_case per variabili e funzioni: nome_completo, calcola_media. Evitate nomi di una sola lettera tranne per contatori brevi (i, j, k nei cicli).



Formattazione di Stringhe con F-String

1

Sintassi Base

Le f-string iniziano con f davanti alle virgolette: f"Ciao {nome}". Le variabili dentro {} vengono sostituite automaticamente con i loro valori.

2

Formattazione Numeri

f"{prezzo:.2f}" mostra 2 decimali, f"{numero:,}" aggiunge separatori delle migliaia. f"{percentuale:.1%}" converte in percentuale con 1 decimale.

3

Espressioni Complesse

Dentro {} si possono inserire calcoli: f"Totale: {prezzo * quantita:.2f}€". Anche chiamate di funzione: f"Oggi è {datetime.now():%d/%m/%Y}".

4

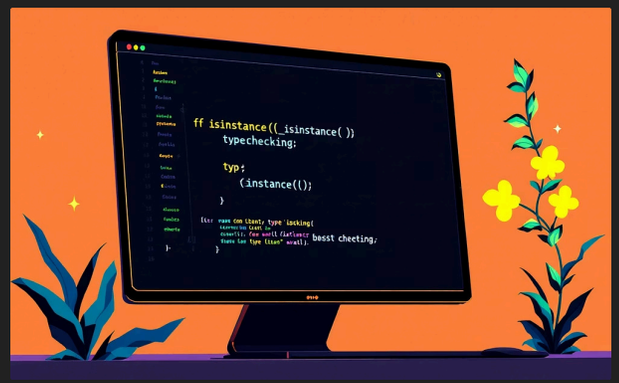
Vantaggi delle F-String

Più leggibili di .format() e % formatting. Più veloci nell'esecuzione. Permettono debug facile: f"{variabile=}" stampa "variabile=valore".

Uso di type() per Verificare Tipi

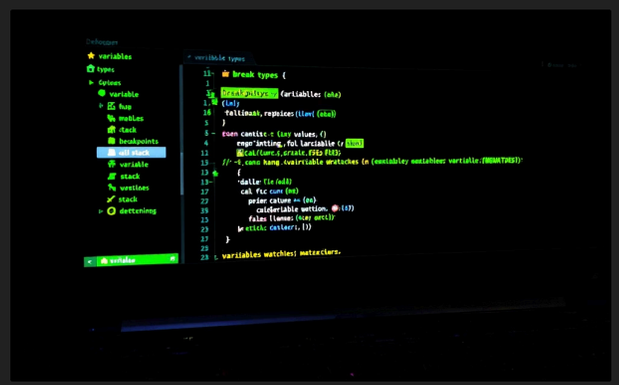
Funzione type()

type(variabile) restituisce il tipo esatto: type(42) è . Utile per debug e per capire cosa contiene una variabile. Si può confrontare: type(x) == int.



isinstance() Preferibile

isinstance(variabile, tipo) è più flessibile: isinstance(3.14, (int, float)) verifica se è numero. Gestisce meglio l'ereditarietà delle classi rispetto a type().



Debug e Validazione

Usate type() per capire errori: se una funzione si aspetta int ma riceve str, type() aiuta a identificarlo. Essenziale durante l'apprendimento per capire il comportamento di Python.

Riepilogo: Dalle Basi alla Pratica

Tipi e Conversioni

Ora conoscete int, float, str, bool e come convertire tra essi. Ricordate: str() funziona sempre, int() e float() richiedono input validi.

Evitare Errori

Divisione per zero, conversioni impossibili, confusione tra = e ==. Ora sapete come prevenire i problemi più comuni!

Operatori Potenti

Operatori aritmetici, logici e di confronto sono i vostri strumenti base. La precedenza conta: usate parentesi quando in dubbio!

Codice Pulito

Nomi significativi, f-string per formattazione, type() per debug. Queste pratiche rendono il codice professionale e manutenibile.