Controllo del Flusso in Python: Condizioni e Cicli

Benvenuti alla lezione fondamentale sui meccanismi di controllo del flusso in Python. Oggi esploreremo come dirigere l'esecuzione del vostro codice attraverso condizioni e iterazioni.

Questa lezione vi fornirà le competenze essenziali per creare programmi dinamici e intelligenti, capaci di prendere decisioni e ripetere operazioni in modo efficiente.

Obiettivi della Lezione

1 Padroneggiare le Strutture Condizionali

Imparare a utilizzare if, elif, else per creare logiche decisionali complesse

2 Comprendere i Cicli di Iterazione

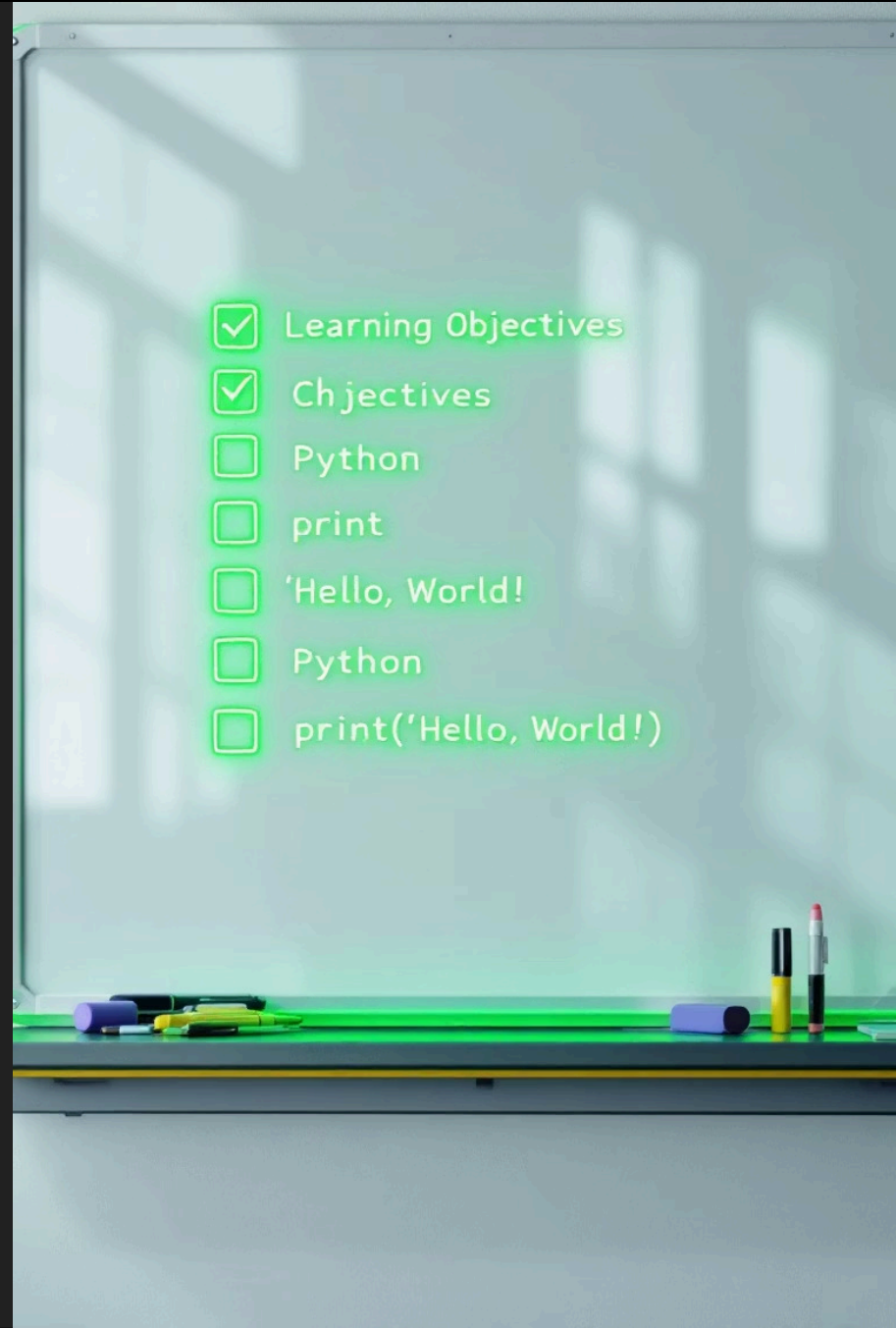
Scoprire for e while per automatizzare operazioni ripetitive sui dati

3 Applicare il Controllo Avanzato

Utilizzare break, continue e pass per gestire il flusso con precisione

4 Sviluppare Buone Pratiche

Scrivere codice leggibile con indentazione corretta e logica chiara



Strutture Condizionali: if, elif, else

Sintassi Base

```
if condizione:
    # codice se vero
elif altra_condizione:
    # codice alternativo
else:
    # codice di default
```

La struttura condizionale permette al programma di scegliere percorsi diversi basandosi su condizioni logiche.

Logica di Valutazione

Python valuta le condizioni dall'alto verso il basso, eseguendo solo il primo blocco che risulta vero. Se nessuna condizione è vera, viene eseguito il blocco else.

Ogni condizione deve restituire un valore booleano (True o False) per funzionare correttamente.

Condizioni Multiple e Annidate

1

Operatori Logici

Combinate condizioni con **and**, **or**, **not** per logiche complesse

2

Condizioni Annidate

Inserite if dentro altri if per creare alberi decisionali approfonditi

3

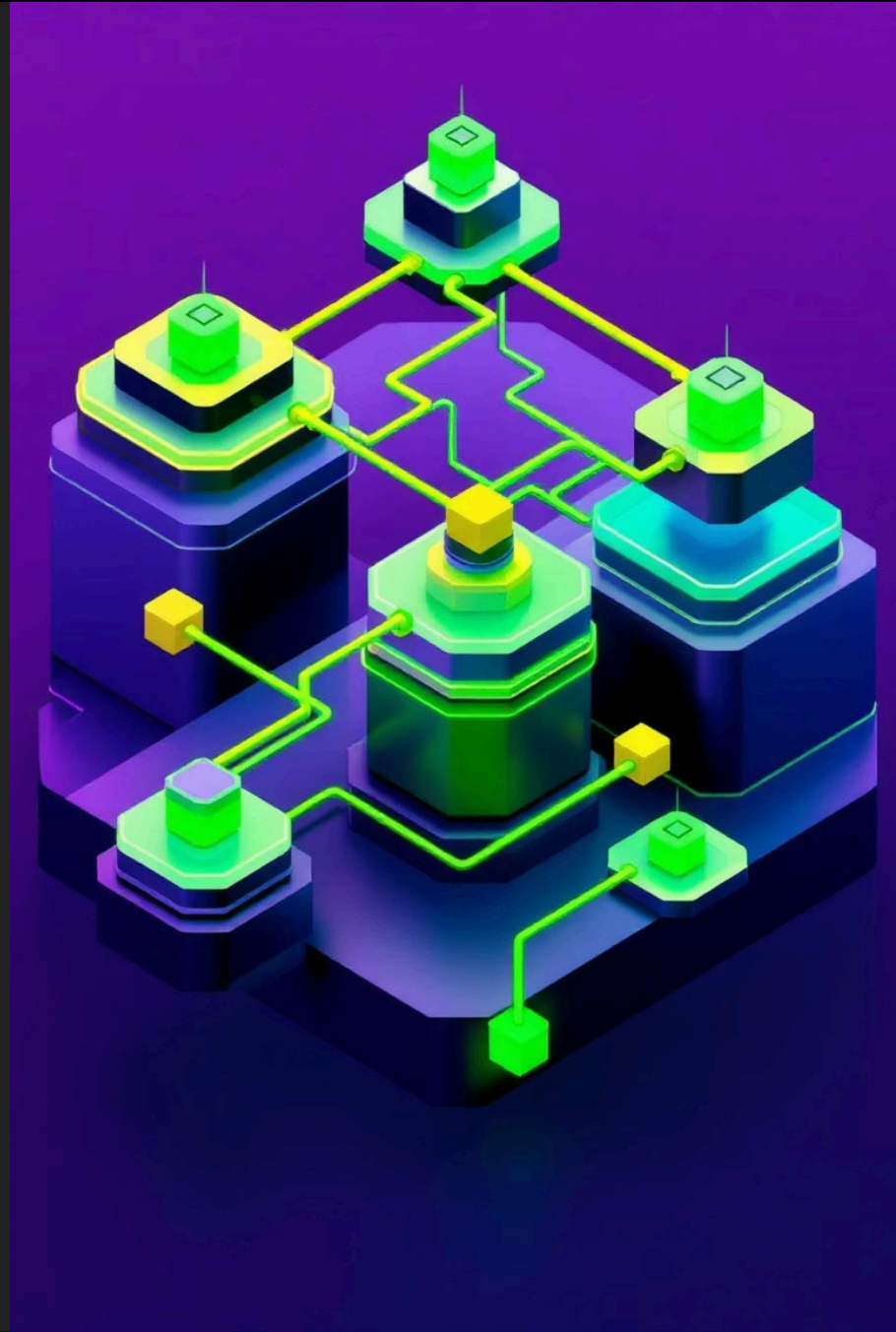
Confronti Multipli

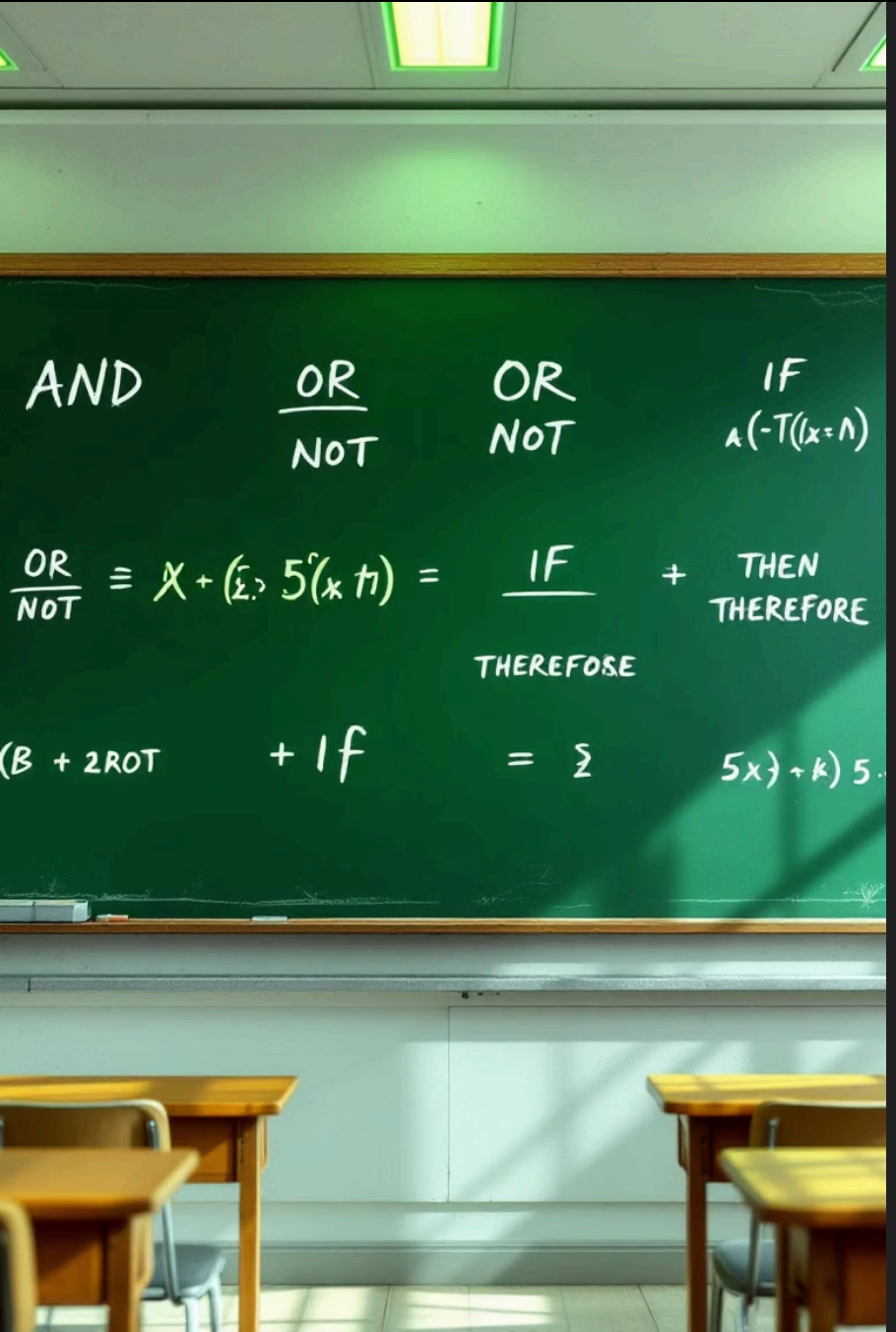
Utilizzate operatori come ==, !=, <, >, <=, >= per confronti numerici e testuali

4

Controllo dell'Appartenenza

Sfruttate **in** e **not in** per verificare la presenza in collezioni





Esercizi di Logica Booleana

Valutazione di Espressioni

Praticate con espressioni come **(5 > 3) and (2 < 4)** per comprendere come Python combina condizioni multiple. Iniziate con esempi semplici e aumentate gradualmente la complessità.

Tabelle di Verità

Create tabelle di verità per operatori logici. Questo esercizio mentale vi aiuterà a prevedere il comportamento del codice prima di eseguirlo.

Scenari Reali

Traducete situazioni quotidiane in logica booleana: "Se piove E ho l'ombrello, esco". Questo collegamento facilita la comprensione astratta.

Cicli For: Iterazione Controllata

1. **Range Numerico**

   for i in range(5) itera da 0 a 4

2. **Liste e Stringhe**

   for item in lista itera su ogni elemento

3. **Dizionari**

   for key in dict itera sulle chiavi

4. **Tuple e Set**

   for elem in collezione gestisce qualsiasi iterabile

Il ciclo for è perfetto quando conoscete in anticipo quante volte dovete ripetere un'operazione o quando dovete processare tutti gli elementi di una collezione. La sua struttura predefinita previene automaticamente i loop infiniti.

Cicli While: Controllo Dinamico

Ripetizione Condizionale

Il while continua finché la condizione rimane vera, ideale per situazioni dove non sapete quante iterazioni servono

Rischio Loop Infinito

Sempre modificare la variabile di controllo dentro il ciclo, altrimenti il programma non si fermerà mai

Condizioni di Guardia

Implementate controlli di sicurezza per evitare iterazioni eccessive che potrebbero bloccare il sistema

Break e Continue: Controllo Preciso

Break: Uscita Immediata

```
for i in range(10):
    if i == 5:
        break
    print(i)
# Stampa: 0, 1, 2, 3, 4
```

Break interrompe completamente il ciclo, passando alla prima istruzione dopo il ciclo stesso.

Continue: Salta Iterazione

```
for i in range(5):
    if i == 2:
        continue
    print(i)
# Stampa: 0, 1, 3, 4
```

Continue salta il resto del codice nell'iterazione corrente e passa direttamente alla successiva.

Pass: Il Placeholder Intelligente

Sintassi Placeholder

Pass è un'istruzione nulla che mantiene la sintassi corretta quando il codice non è ancora implementato. Essenziale durante lo sviluppo incrementale.

Sviluppo Strutturato

Utilizzate pass per creare la struttura del programma prima di implementare la logica. Questo approccio migliora la pianificazione del codice.

Debug e Testing

Pass permette di testare parti del codice disabilitando temporaneamente sezioni specifiche senza causare errori di sintassi.

Range: Personalizzazione Avanzata

range(stop)

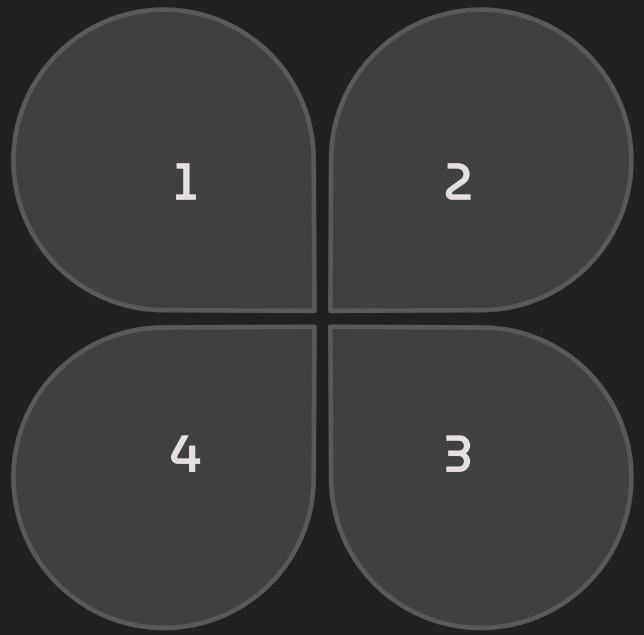
Da 0 a stop-1

- range(5) = 0,1,2,3,4
- Forma più semplice

range(start, stop)

Da start a stop-1

- range(2, 7) = 2,3,4,5,6
- Controllo punto iniziale



Step Negativi

Per sequenze inverse

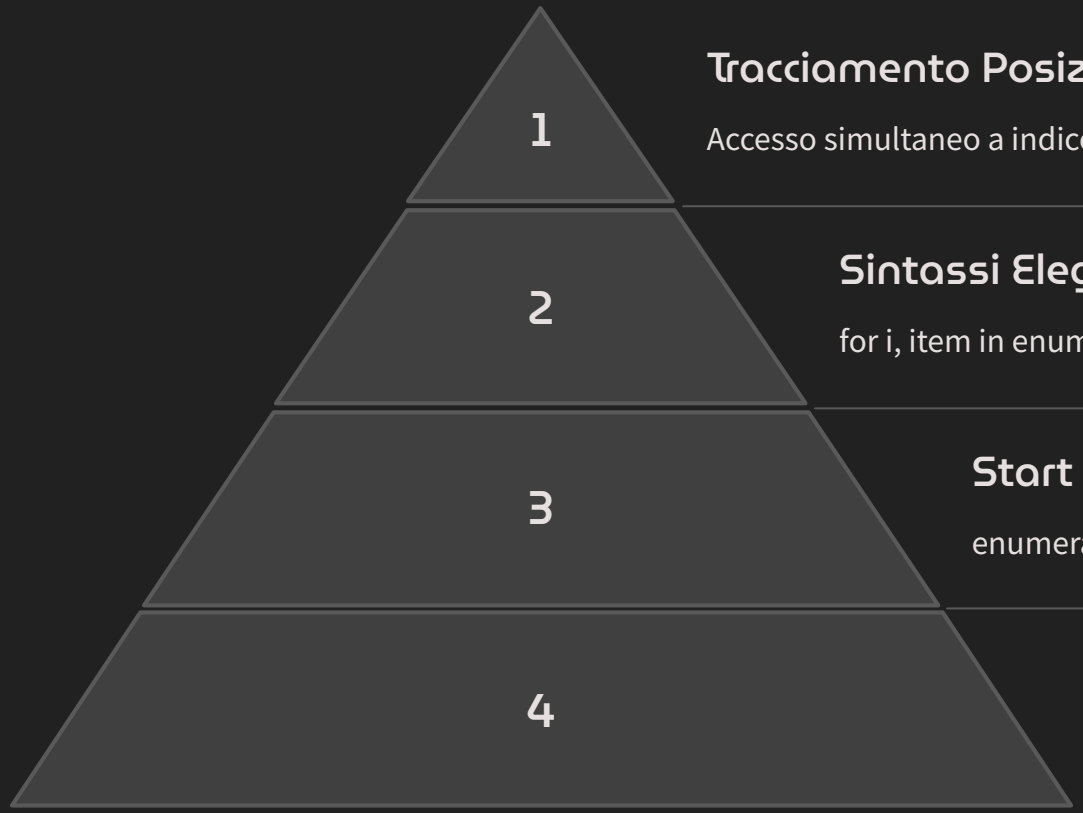
- range(10, 0, -1) = 10,9,8...1
- Countdown e reverse

range(start, stop, step)

Con incremento personalizzato

- range(0, 10, 2) = 0,2,4,6,8
- Salti e decrementi

Enumerate: Indici e Valori Insieme



Tracciamento Posizione

Accesso simultaneo a indice e valore

Sintassi Elegante

for i, item in enumerate(lista)

Start Personalizzato

enumerate(lista, start=1) inizia da 1

Caso d'Uso

Perfetto per numerare output e referenziare posizioni

Enumerate è particolarmente utile quando dovete stampare liste numerate o quando necessitate sia della posizione che del valore di ogni elemento. Evita la necessità di mantenere contatori manuali, riducendo gli errori e migliorando la leggibilità del codice.

Nesting: Strutture Intrecciate

1

Cicli in Condizioni

For loop all'interno di blocchi if per processare dati condizionalmente

2

Condizioni in Cicli

If statement dentro for loop per filtrare elementi durante iterazione

3

Cicli Annidati

For dentro for per elaborare matrici e strutture bidimensionali

4

Complessità Controllata

Limitare i livelli di nesting per mantenere codice comprensibile

Indentazione: La Struttura del Codice

Quattro Spazi

Standard Python per ogni livello di indentazione, garantisce consistenza

Collaborazione Efficace

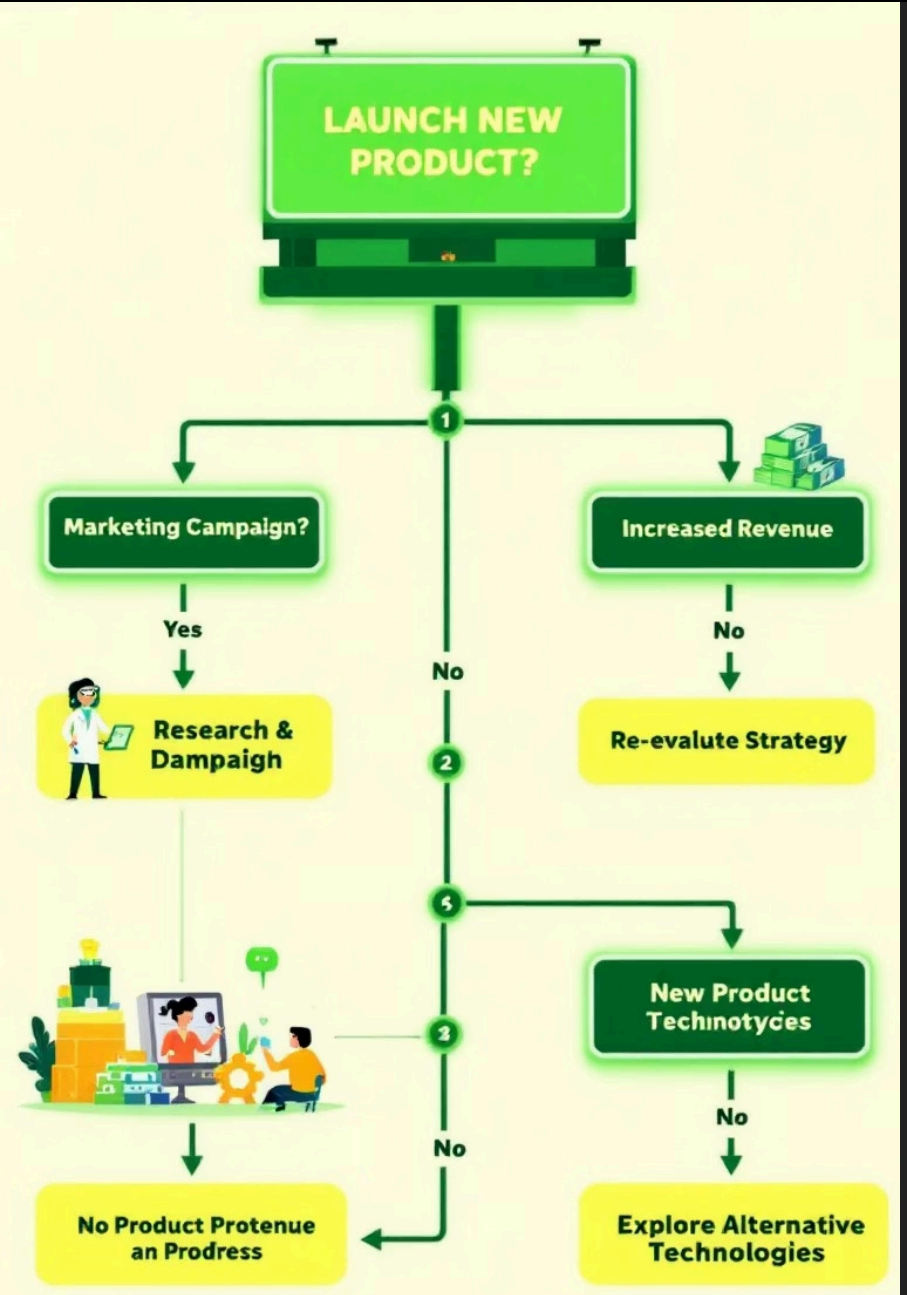
Codice uniformemente indentato facilita il lavoro di squadra

Leggibilità Visiva

Indentazione corretta rende la struttura logica immediatamente comprensibile

Prevenzione Errori

IndentationError avvisa di problemi strutturali prima dell'esecuzione



Simulazione di Logiche Decisionali

Sistema di Login

Verificare username e password con condizioni annidate per gestire accesso, tentativi falliti e blocco account

Calcolatrice Intelligente

Implementare operazioni matematiche con controllo divisione per zero e validazione input utente

Sistema di Voti

Calcolare medie, assegnare lettere di voto e determinare promozioni basandosi su soglie multiple

E-commerce Logic

Gestire carrello, sconti, spedizione e pagamenti con controlli di inventario e validazioni

Esempi Pratici: Numeri, Stringhe e Condizioni

42

Numeri Primi

Verificare i numeri primi usando cicli for e if: controlla se un numero è divisibile solo per 1 e per se stesso

100

Percentuali

Calcolare percentuali di completamento: usa f-string per formattare e confronta il risultato con valori obiettivo

256

Potenze di 2

Creare una sequenza di potenze di 2: genera i numeri 2, 4, 8, 16... fino a 256 usando cicli o list comprehension

ABC

Stringhe

Controllare testi: verifica se una stringa contiene solo lettere, ha lunghezza sufficiente e inizia con maiuscola

Questi esempi mostrano come usare il controllo di flusso con dati reali. Esercitarsi con questi modelli aiuta a combinare if, for e while con i metodi di base, creando le competenze necessarie per risolvere problemi di programmazione comuni.