# Esercizi Pratici di Programmazione per Principianti

Benvenuti a questo corso pratico di programmazione! Oggi esploreremo 15 esercizi fondamentali che vi aiuteranno a costruire solide basi nella programmazione. Questi esercizi coprono concetti essenziali come condizioni, cicli, liste e debugging.

Ogni esercizio è progettato per insegnare specifiche competenze di programmazione, partendo da problemi semplici fino ad arrivare a sfide più complesse. Attraverso esempi pratici e spiegazioni dettagliate, imparerete a pensare come programmatori.

# Determinare se un Numero è Pari o Dispari

### Input dell'utente

Richiedere all'utente di inserire un numero intero utilizzando la funzione input()

### Operatore modulo

Utilizzare l'operatore % per calcolare il resto della divisione per 2

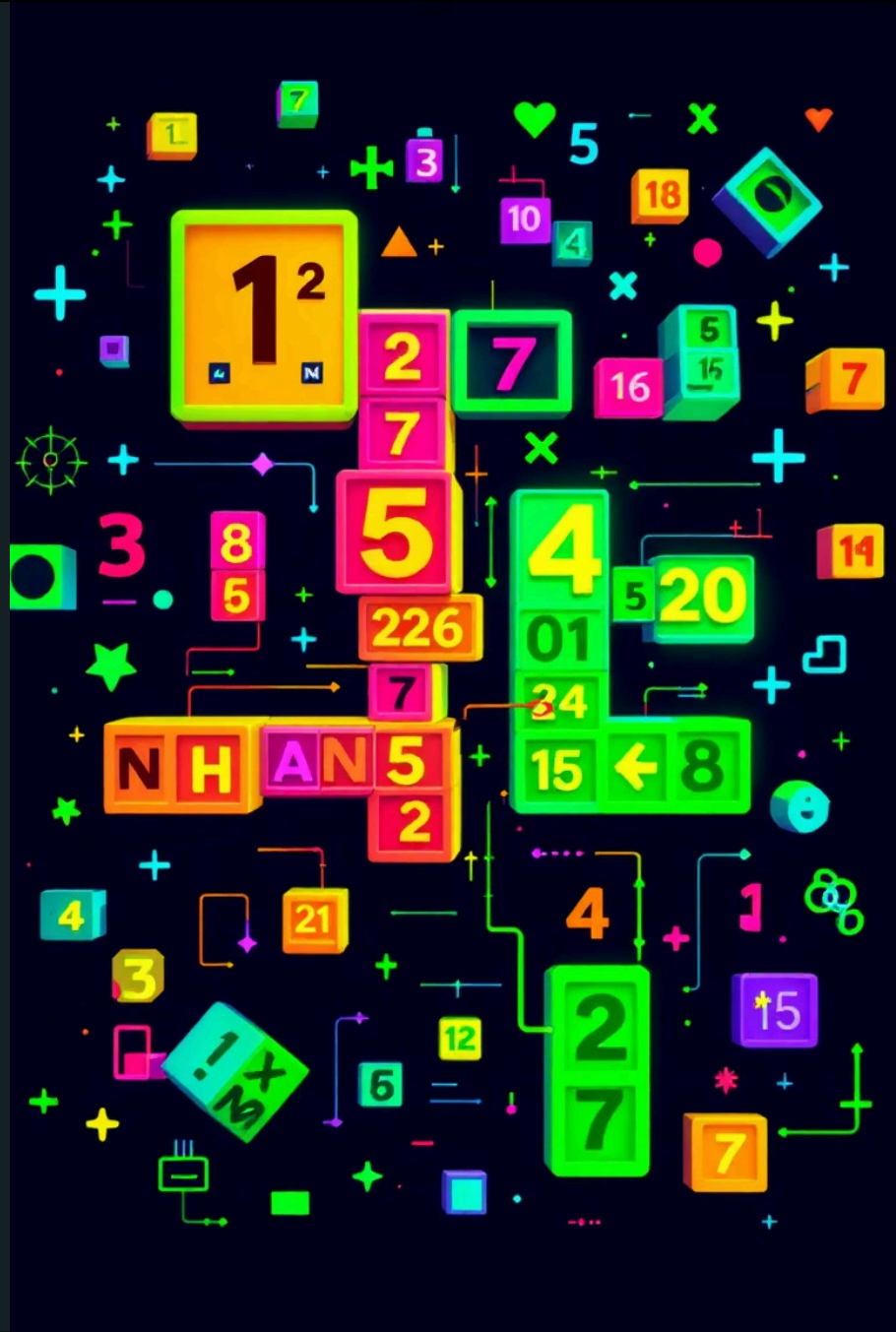
### Struttura condizionale

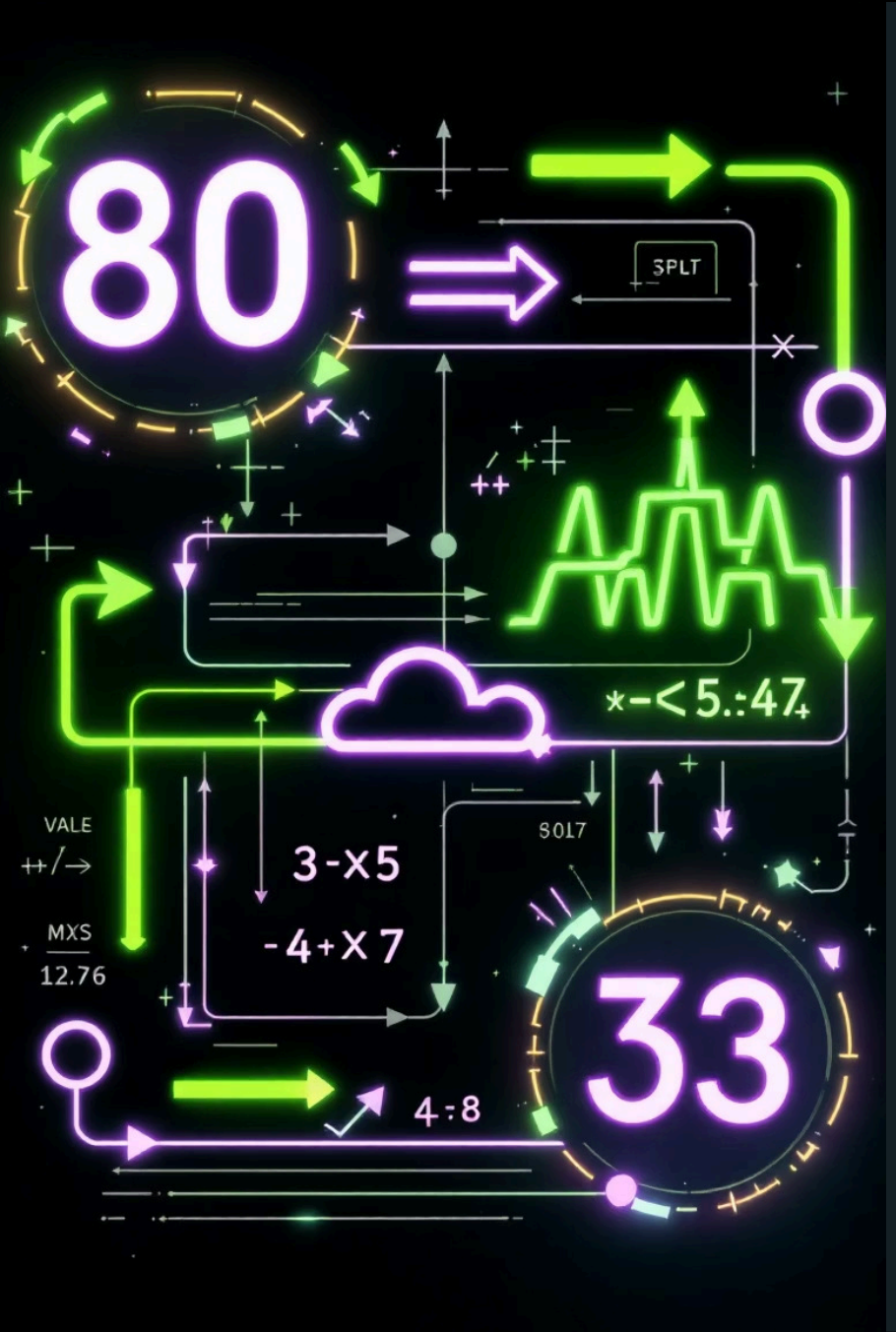
Implementare if-else per verificare se il resto è 0 (pari) o 1 (dispari)

### Output del risultato

Stampare un messaggio chiaro che indichi se il numero è pari o dispari

Questo esercizio introduce i concetti fondamentali di input utente, operazioni matematiche e strutture condizionali. L'operatore modulo (%) è particolarmente importante perché restituisce il resto di una divisione, permettendo di identificare facilmente numeri pari e dispari.






# Calcolo del Massimo tra Tre Numeri

### Acquisizione dati

Richiedere tre numeri dall'utente e convertirli in formato numerico

### Confronti multipli

Utilizzare operatori di confronto per determinare il numero maggiore

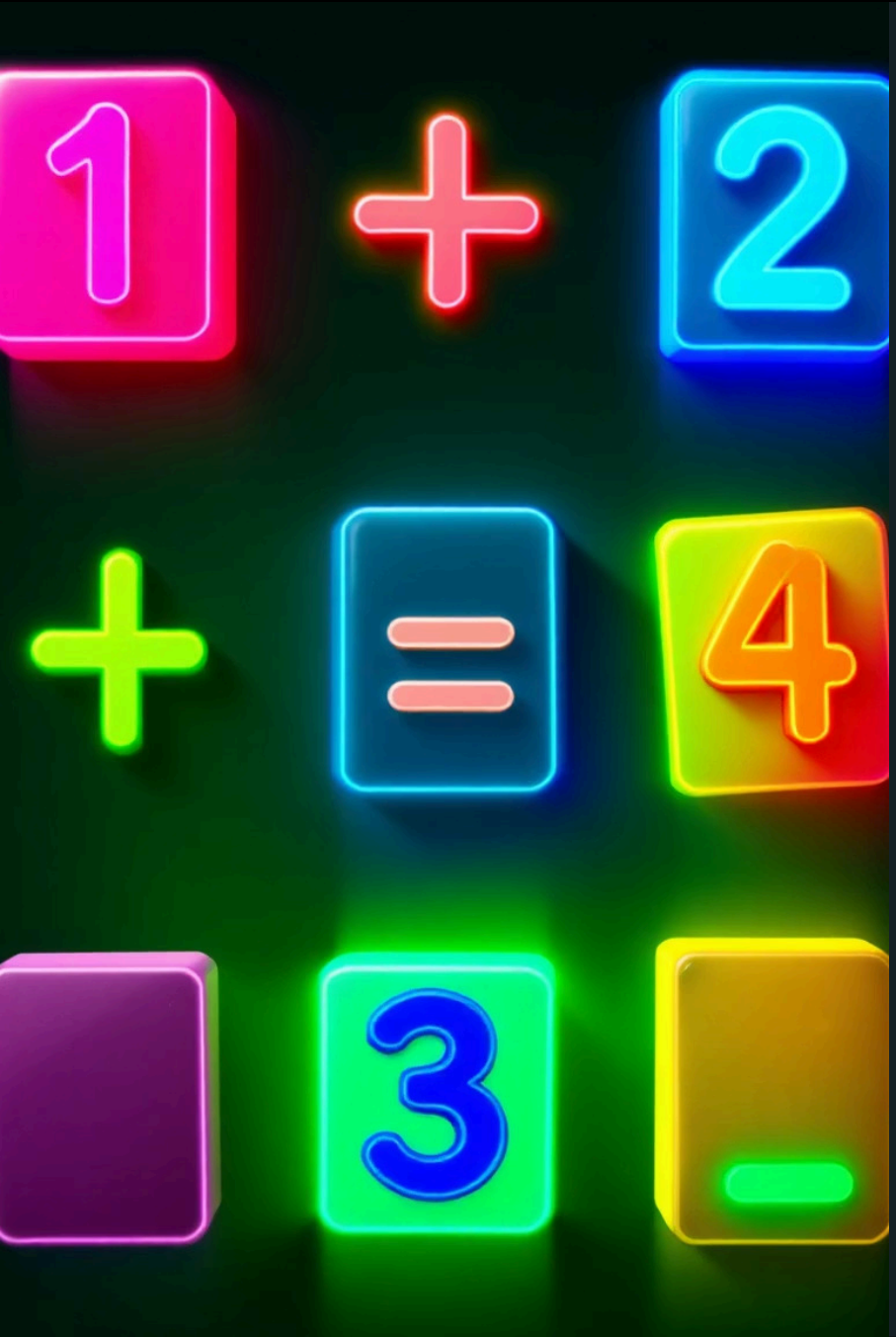
### Identificazione massimo

Implementare la logica per trovare il valore più grande tra i tre

### Comunicazione risultato

Mostrare chiaramente quale numero è il massimo

Questo esercizio sviluppa le competenze nel confronto di valori e nell'uso di condizioni annidate. È importante considerare tutti i casi possibili, inclusi quelli in cui due o tre numeri sono uguali. L'esercizio può essere risolto sia con if-elif-else che utilizzando la funzione max() integrata di Python.



# Stampa dei Primi N Numeri

### Definizione dell'intervallo

Richiedere all'utente quanti numeri stampare e validare l'input ricevuto

### Implementazione del ciclo

Utilizzare un ciclo for con range() per iterare attraverso i numeri desiderati

### Formattazione output

Stampare ogni numero in modo chiaro, considerando se mostrarli su righe separate o sulla stessa riga

Questo esercizio introduce il concetto fondamentale dei cicli for e della funzione range(). È importante spiegare come range(n) genera numeri da 0 a n-1, e come range(1, n+1) può essere usato per numeri da 1 a n. L'esercizio può essere esteso per includere formattazione personalizzata dell'output.

# Somma dei Numeri da 1 a N con For

## Approccio algoritmico

Inizializzare una variabile somma a zero e utilizzare un ciclo for per aggiungere ogni numero dall'1 a N. Questo metodo insegna l'accumulo progressivo di valori.

- Variabile accumulo inizializzata
- Ciclo for con range appropriato
- Operazione di somma iterativa

## Validazione matematica

Confrontare il risultato ottenuto con la formula matematica N*(N+1)/2 per verificare la correttezza dell'algoritmo implementato.

- Formula matematica diretta
- Confronto dei risultati
- Verifica dell'accuratezza

Questo esercizio combina programmazione e matematica, introducendo il concetto di variabili accumulatrici. È un ottimo esempio per spiegare come i computer possano risolvere problemi che hanno anche soluzioni matematiche dirette, e permette di discutere l'efficienza algoritmica.

# Uso del Ciclo While per Contare al Contrario

### Inizializzazione

Impostare la variabile contatore al valore di partenza inserito dall'utente

### Condizione del ciclo

Verificare che il contatore sia maggiore di zero per continuare l'iterazione

### Decremento

Ridurre il contatore di 1 per avvicinarsi alla condizione di terminazione

### Stampa valore

Mostrare il valore corrente del contatore prima di decrementarlo

Il ciclo while è particolarmente utile quando non conosciamo esattamente quante iterazioni saranno necessarie. In questo esercizio, pur conoscendo il numero di iterazioni, impariamo la struttura del while e l'importanza di modificare la variabile di controllo per evitare cicli infiniti. È fondamentale ricordare di decrementare il contatore!


# Esercizio: Tabellina del Numero Inserito

### Input e validazione

Richiedere all'utente di inserire un numero e verificare che sia un intero valido. Gestire eventuali errori di input con messaggi informativi.

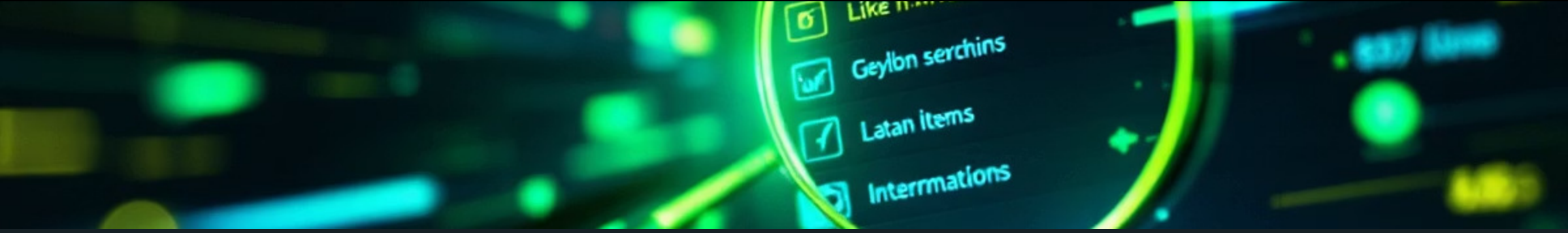
### Generazione tabellina

Utilizzare un ciclo for da 1 a 10 per calcolare e mostrare ogni moltiplicazione. Formattare l'output in modo leggibile e ordinato.

### Presentazione risultati

Stampare ogni riga della tabellina nel formato "N x i = risultato" per una comprensione immediata del calcolo eseguito.

Questo esercizio combina cicli, operazioni matematiche e formattazione dell'output. È un'ottima opportunità per discutere la presentazione dei dati e l'importanza di output chiari e ben formattati. Può essere esteso per permettere all'utente di scegliere fino a quale numero moltiplicare.




# Ricerca in una Lista: Presenza Elemento

### Creazione lista

Definire una lista di elementi o permettere all'utente di inserirli

### Elemento da cercare

Richiedere all'utente quale elemento vuole trovare nella lista

### Algoritmo di ricerca

Implementare la ricerca usando 'in' operator o ciclo manuale

### Risultato ricerca

Comunicare se l'elemento è presente e eventualmente la sua posizione

Questo esercizio introduce le liste e gli algoritmi di ricerca. È importante mostrare sia l'operatore 'in' di Python (più semplice) che l'implementazione manuale con un ciclo (più educativa). La ricerca manuale permette di trovare anche la posizione dell'elemento e di contare le occorrenze multiple.

# Calcolo Media e Numero Massimo da Lista Input

### Acquisizione dati

Permettere all'utente di inserire una serie di numeri, terminando con un valore speciale o contando le entries

### Calcolo media

Sommare tutti i valori e dividere per il numero totale di elementi inseriti

### Identificazione massimo

Trovare il valore più grande utilizzando la funzione max() o un confronto iterativo

### Presentazione statistiche

Mostrare media e massimo con formattazione appropriata e decimali limitati

Questo esercizio combina manipolazione di liste, operazioni matematiche e gestione dell'input utente. È importante gestire il caso di lista vuota e formattare correttamente i numeri decimali. L'esercizio può essere esteso per includere altre statistiche come minimo, mediana e deviazione standard.


# Contatore con Ciclo e Condizione

### Configurazione parametri

Definire valore iniziale, finale e incremento del contatore

### Applicazione condizioni

Implementare filtri per contare solo elementi che soddisfano criteri specifici

### Esecuzione ciclo

Utilizzare while o for per iterare e verificare condizioni ad ogni passo

### Visualizzazione progresso

Mostrare il conteggio in tempo reale e il risultato finale

Questo esercizio insegna l'uso di contatori condizionali, utili per statistiche e analisi dati. Esempi pratici includono contare numeri pari in un intervallo, caratteri specifici in una stringa, o elementi che soddisfano criteri complessi. È fondamentale per comprendere l'elaborazione selettiva dei dati.

# Esercizio: Identificazione Numeri Primi

### Definizione matematica

Numero divisibile solo per 1 e se stesso

### Controllo divisori

Testare divisibilità da 2 fino alla radice quadrata

### Implementazione ciclo

Utilizzare for con range per verificare ogni possibile divisore

### Ottimizzazioni

Gestire casi speciali come 2 e numeri pari

### Validazione input

Verificare che il numero sia positivo e maggiore di 1

L'identificazione dei numeri primi è un classico problema algoritmico che introduce concetti matematici avanzati. L'algoritmo base verifica se esistono divisori tra 2 e n-1, ma può essere ottimizzato fermandosi alla radice quadrata di n. Questo esercizio insegna l'importanza dell'efficienza algoritmica e delle ottimizzazioni matematiche.

# Stampa Solo Numeri Pari da 1 a 100

### Metodo range con step

Utilizzare range(2, 101, 2) per generare direttamente numeri pari

### Metodo con filtro modulo

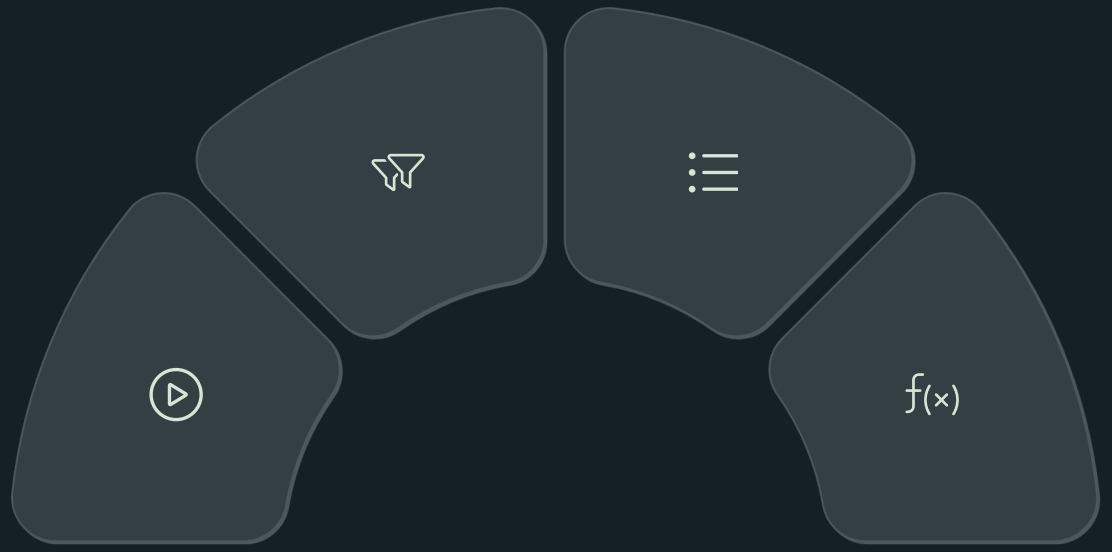
Ciclare da 1 a 100 e stampare solo se n % 2 == 0

### Metodo list comprehension

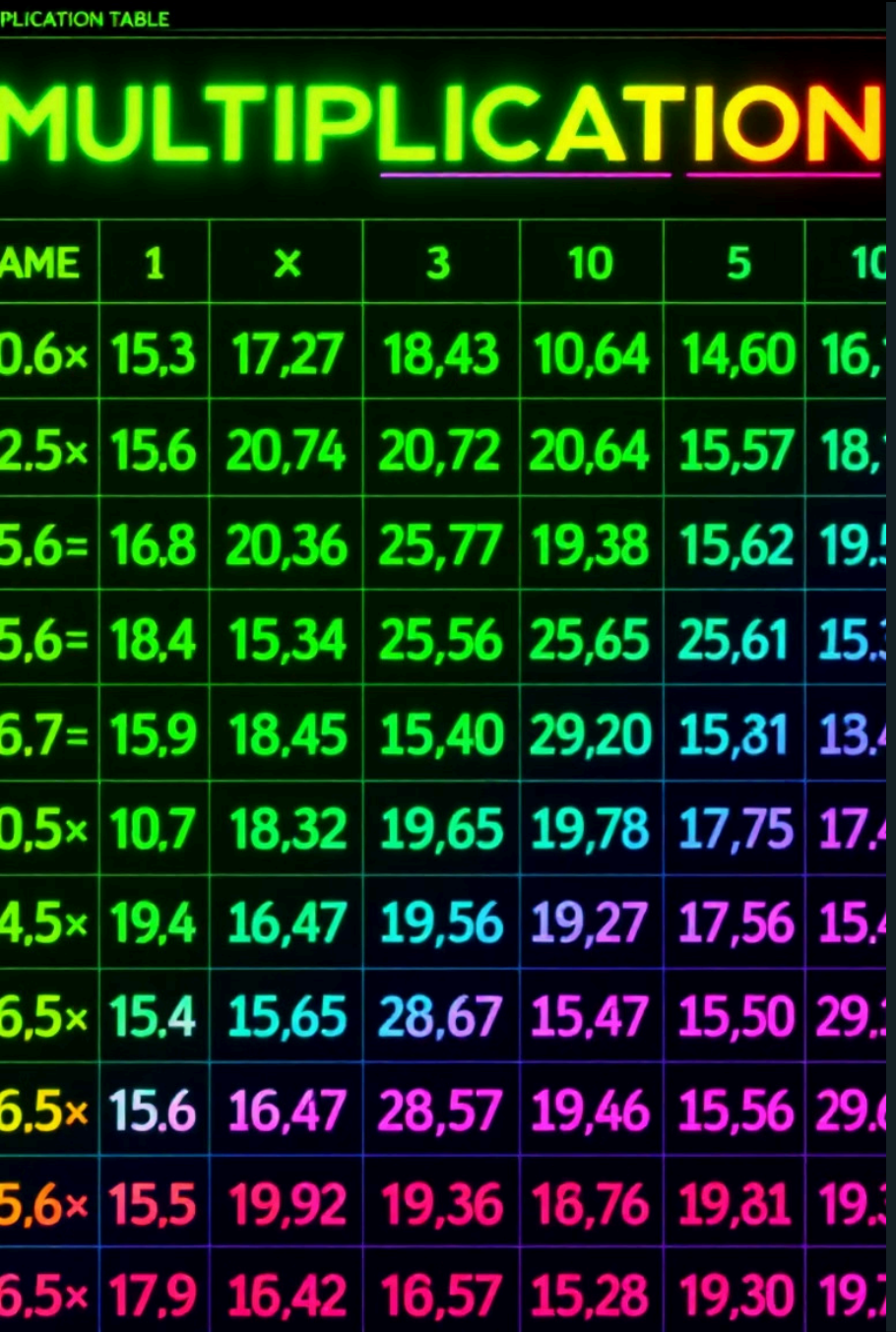
Creare lista con [n for n in range(1, 101) if n % 2 == 0]

### Metodo con funzione

Implementare una funzione is_even() per maggiore leggibilità




Questo esercizio mostra diverse approcci per risolvere lo stesso problema, evidenziando come Python offra multiple soluzioni. È importante discutere l'efficienza di ogni metodo: range con step è più efficiente perché genera solo i numeri necessari, mentre il filtro con modulo deve verificare ogni numero.



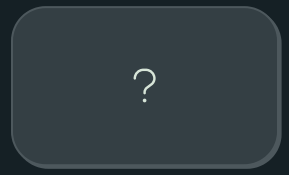

# Nested Loop: Tabella Moltiplicazioni

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **1** | 1 | 2 | 3 | 4 | 5 |
| **2** | 2 | 4 | 6 | 8 | 10 |
| **3** | 3 | 6 | 9 | 12 | 15 |
| **4** | 4 | 8 | 12 | 16 | 20 |

I cicli annidati (nested loops) sono fondamentali per elaborare strutture bidimensionali. Il ciclo esterno controlla le righe mentre quello interno gestisce le colonne. È cruciale comprendere come le variabili di entrambi i cicli interagiscono per produrre ogni cella della tabella.

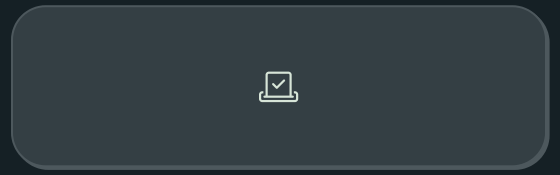
La formattazione dell'output richiede attenzione speciale per allineare correttamente i numeri in colonne. Utilizzare string formatting o la funzione format() per garantire una presentazione professionale dei risultati. L'esercizio può essere esteso per includere intestazioni di riga e colonna.

# Validazione Input con While



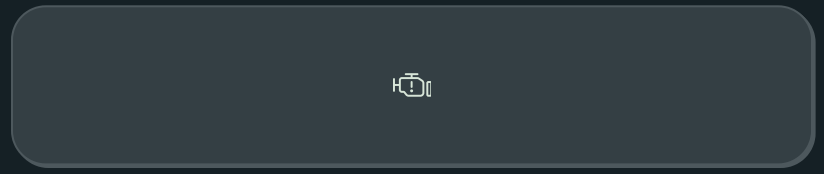
### Richiesta input iniziale

Presentare chiaramente cosa si aspetta dall'utente



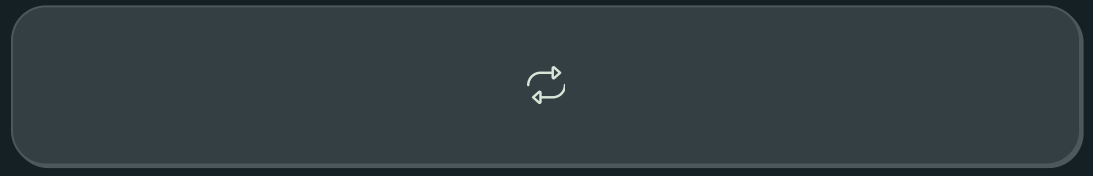
### Validazione criteri

Verificare tipo, range e formato dell'input ricevuto



### Gestione errori

Mostrare messaggi di errore specifici e informativi



### Richiesta ripetuta

Continuare finché l'input non è valido

La validazione dell'input è essenziale per creare programmi robusti. Il ciclo while permette di ripetere la richiesta finché l'utente non fornisce dati validi. È importante gestire diversi tipi di errore: ValueError per conversioni numeriche fallite, range errors per valori fuori dall'intervallo accettabile, e format errors per stringhe malformate.

Implementare messaggi di errore chiari e specifici migliora significativamente l'esperienza utente. Considerare anche un limite massimo di tentativi per evitare cicli infiniti in caso di input persistentemente errati.

# Debugging: Infinite Loop e Condizioni Errate

3

Tipi di errori comuni

Sintassi, logica e runtime errors

5

Tecniche debugging

Print statements, debugger, code review

∞

Loop infiniti

Condizioni che non cambiano mai

1

Errore off-by-one

Problema più frequente nei cicli

Il debugging è una competenza fondamentale che ogni programmatore deve sviluppare. Gli infinite loop sono spesso causati da condizioni di terminazione che non vengono mai soddisfatte o variabili di controllo che non vengono modificate correttamente all'interno del ciclo.

Tecniche efficaci includono l'uso di print statements strategici per tracciare il flusso del programma, l'utilizzo del debugger integrato nell'IDE, e la pratica del "rubber duck debugging" - spiegare il codice riga per riga ad alta voce. Imparare a leggere e interpretare i messaggi di errore è altrettanto cruciale per una risoluzione rapida dei problemi.