Il linguaggio Python

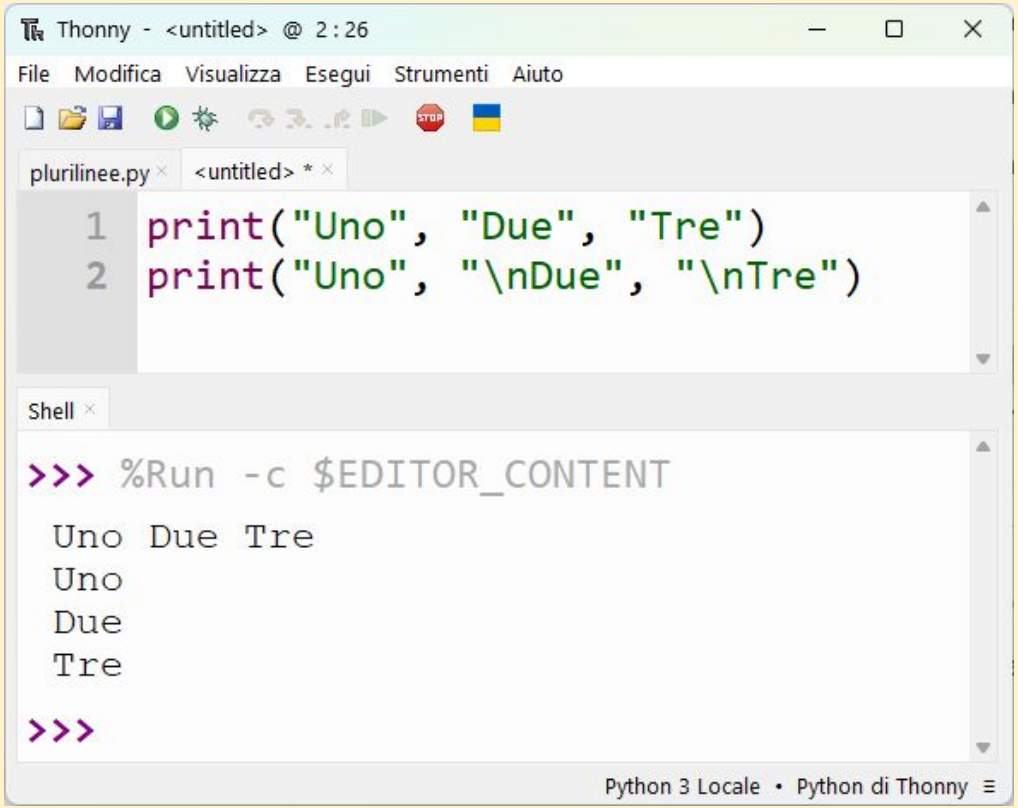
Istruzione di output

- In Python, l'output è gestito principalmente dalla funzione `print()`, che è molto più flessibile e "intelligente" rispetto al `printf` del linguaggio C.

- Non hai bisogno di specificatori di formato obbligatori (come `%d` o `%f`) perché Python capisce da solo il tipo di dato.

Istruzione di output

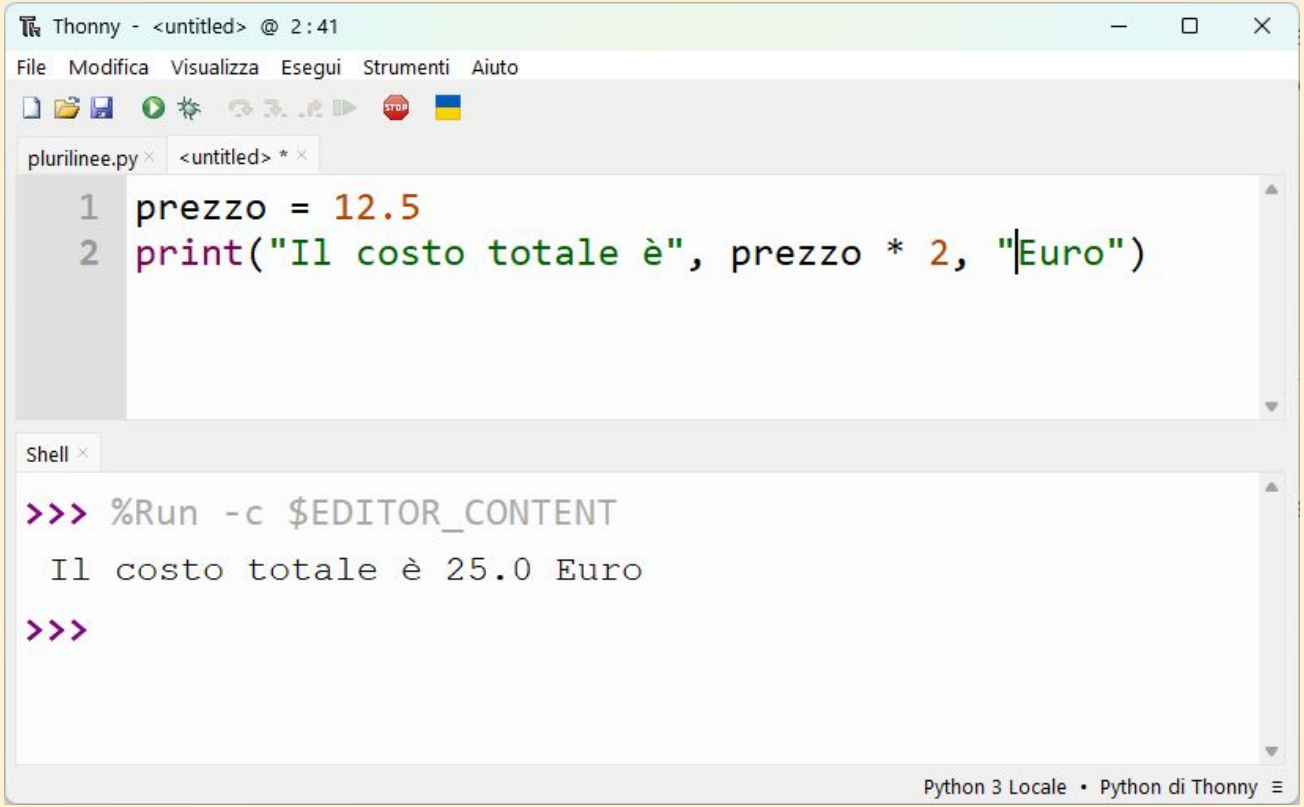
Istruzione di output



Il carattere di escape **“\n”** consente di andare a capo

Il carattere di escape **“\t”** consente di tabulare

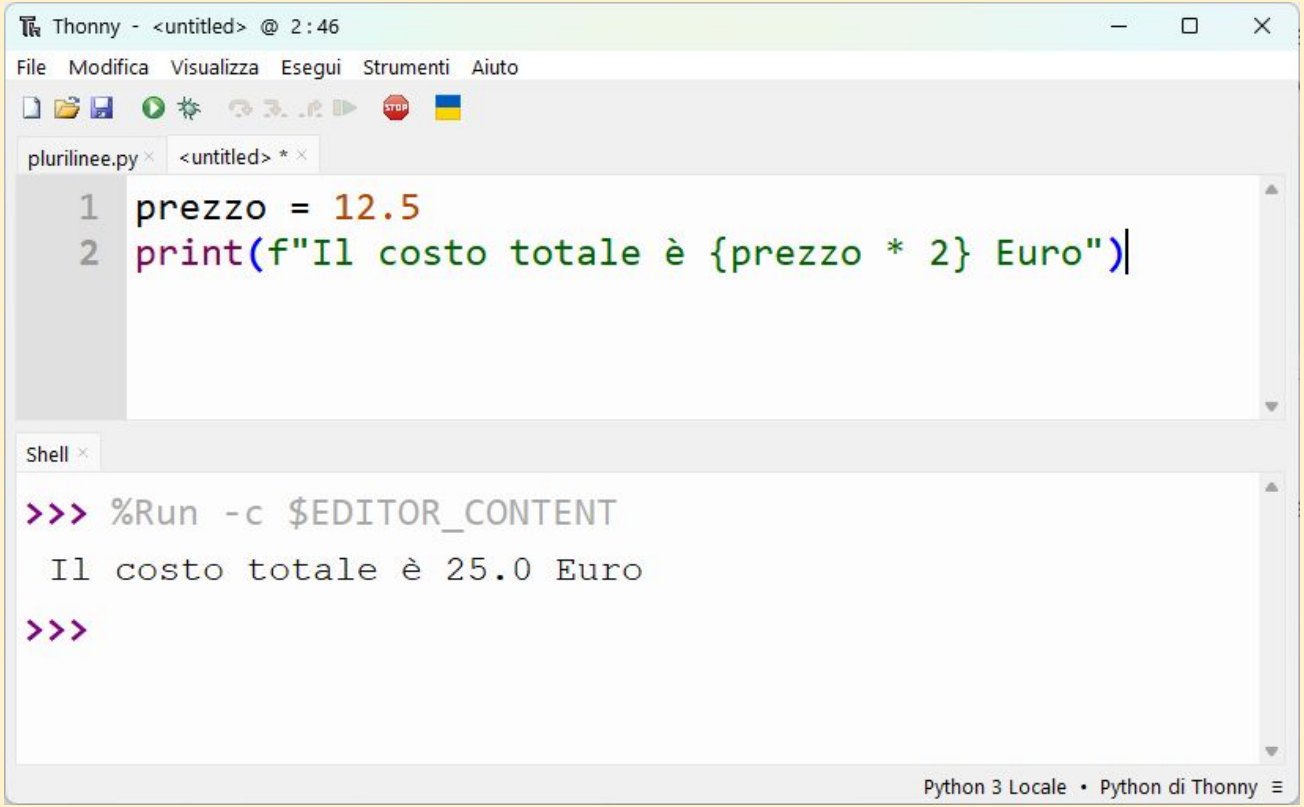
Output di variabili



La funzione `print()` e i suoi parametri

La funzione base può accettare più argomenti separati dalla virgola. Python aggiunge automaticamente uno spazio tra di loro.

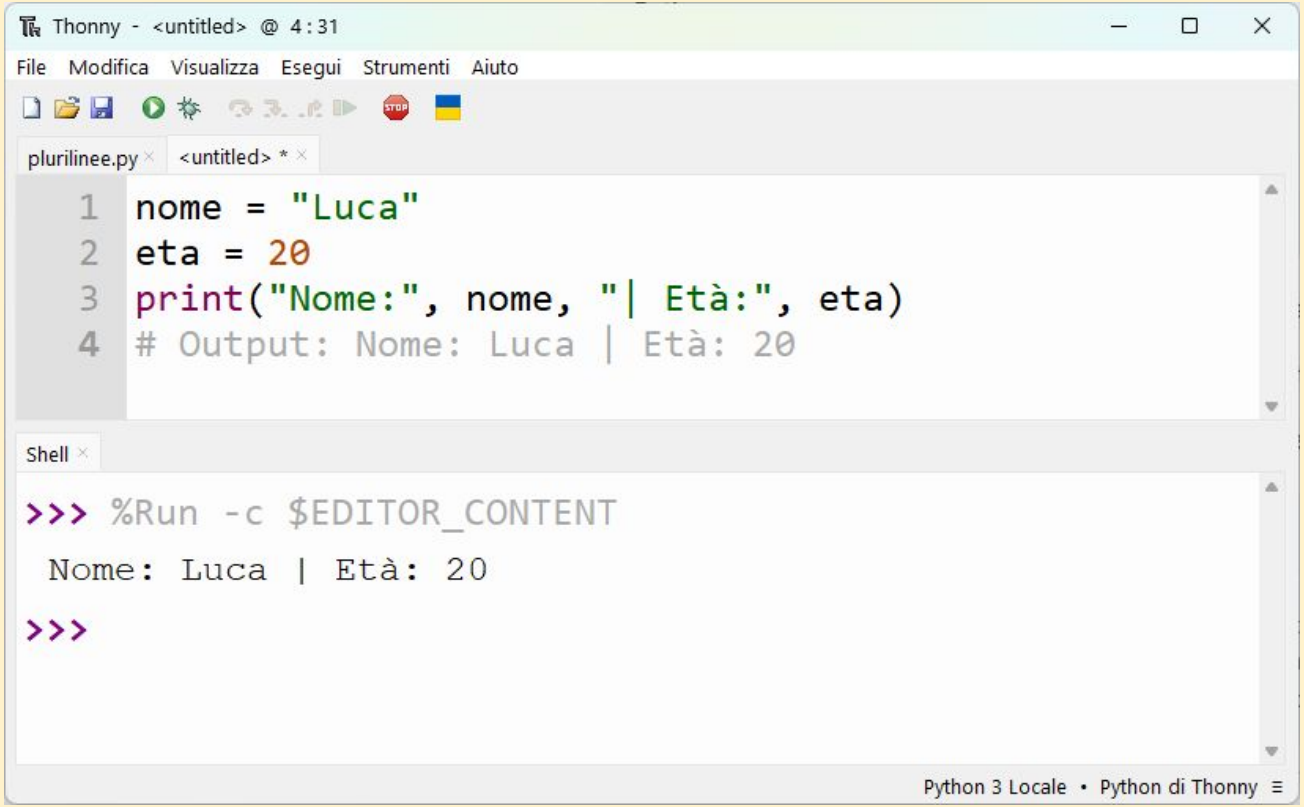
Output di variabili



f-strings (Il metodo moderno)

Introdotte in Python 3.6, le **f-strings** sono il modo più rapido e leggibile per inserire variabili e calcoli dentro una stringa. Basta mettere una `f` prima delle virgolette e usare le graffe `{}`.

Output


Thonny - <untitled> @ 4 : 31
File Modifica Visualizza Esegui Strumenti Aiuto
plurilinee.py
<untitled> *
1 nome = "Luca"
2 eta = 20
3 print("Nome:", nome, "| Età:", eta)
4 # Output: Nome: Luca | Età: 20
Shell
>>> %Run -c $EDITOR_CONTENT
Nome: Luca | Età: 20
>>>
Python 3 Locale • Python di Thonny

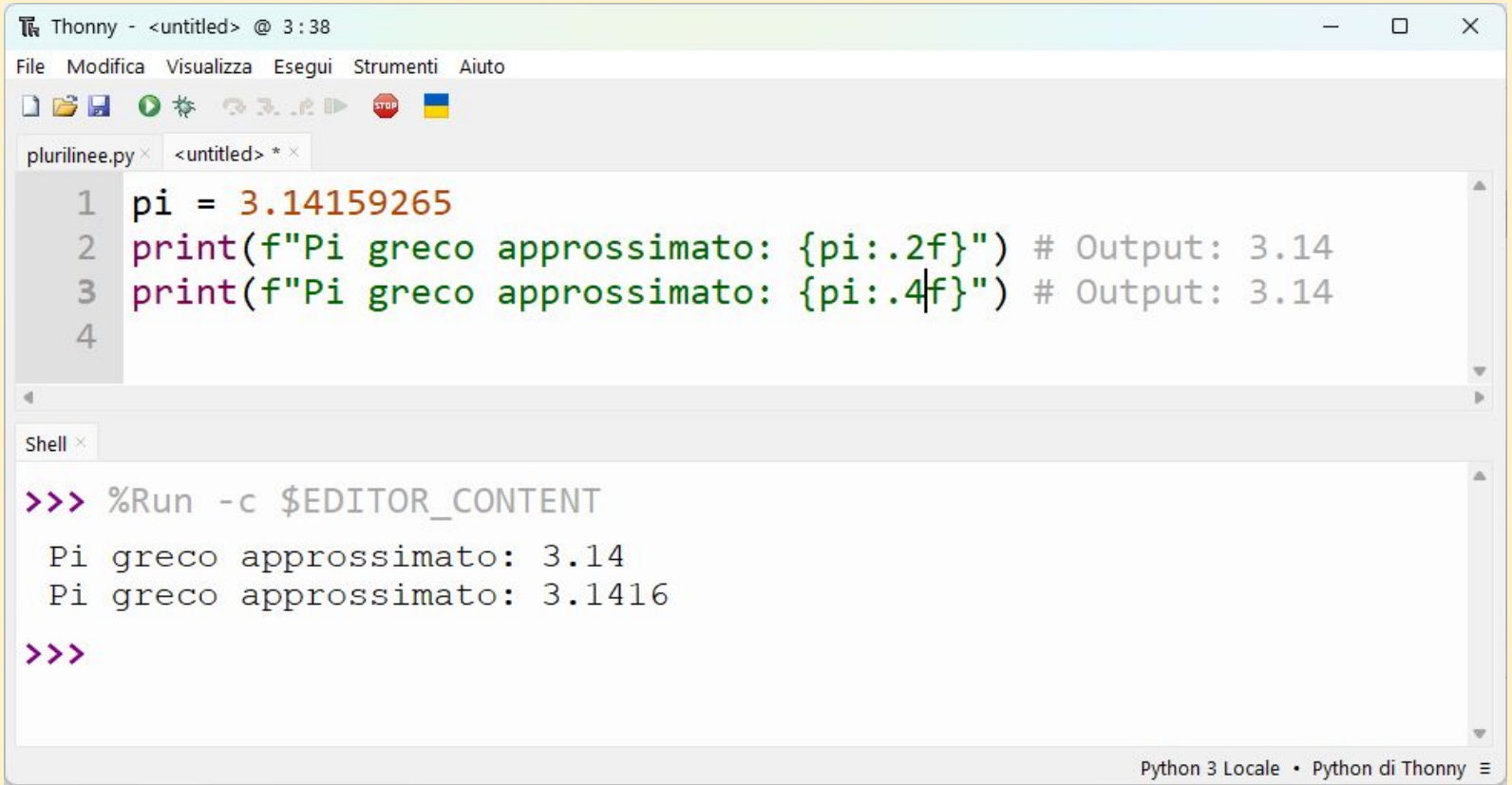
Output

Formattazione dei numeri (Precisione)

Proprio come in C, puoi controllare i decimali e la spaziatura. All'interno delle graffe si usa la sintassi `{variabile:formattazione}`.

- **Decimali:** `:.2f` (mostra 2 decimali).
- **Padding (Spazi):** `:10` (riserva 10 spazi, utile per le tabelle).

Output


Thonny - <untitled> @ 3 : 38
File Modifica Visualizza Esegui Strumenti Aiuto
plurilinee.py
<untitled> *
1 pi = 3.14159265
2 print(f"Pi greco approssimato: {pi:.2f}") # Output: 3.14
3 print(f"Pi greco approssimato: {pi:.4f}") # Output: 3.14
4
Shell
>>> %Run -c $EDITOR_CONTENT
Pi greco approssimato: 3.14
Pi greco approssimato: 3.1416
>>>
Python 3 Locale • Python di Thonny

Separatore


Thonny - <untitled> @ 3 : 69
File Modifica Visualizza Esegui Strumenti Aiuto
plurilinee.py
<untitled> *
1 print("Uno", "Due", "Tre", sep=" - ") # Output: Uno - Due - Tre
2 print("Caricamento...", end="") # Non va a capo
3 print("Fatto!") # Output: Caricamento...Fatto!
4
Shell
>>> %Run -c $EDITOR_CONTENT
Uno - Due - Tre
Caricamento...Fatto!
>>>
Python 3 Locale • Python di Thonny

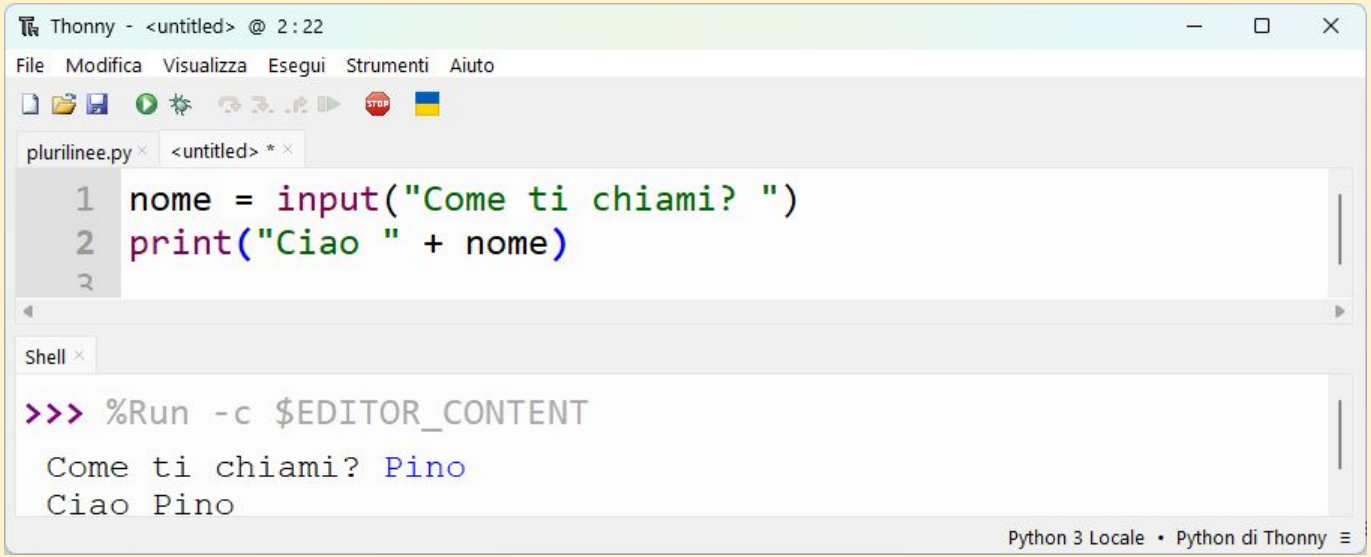
Output - tabella

Funzionalità	Codice Python	Risultato / Scopo
Separatore	`sep=","`	Cambia lo spazio tra gli elementi
Fine riga	`end=" "`	Evita di andare a capo dopo la stampa
f-string	`f"{var}"`	Inserisce variabili direttamente nel testo
Precisione	`f"{var:.2f}"`	Taglia i decimali (come `%.2f` in C)

Input in python

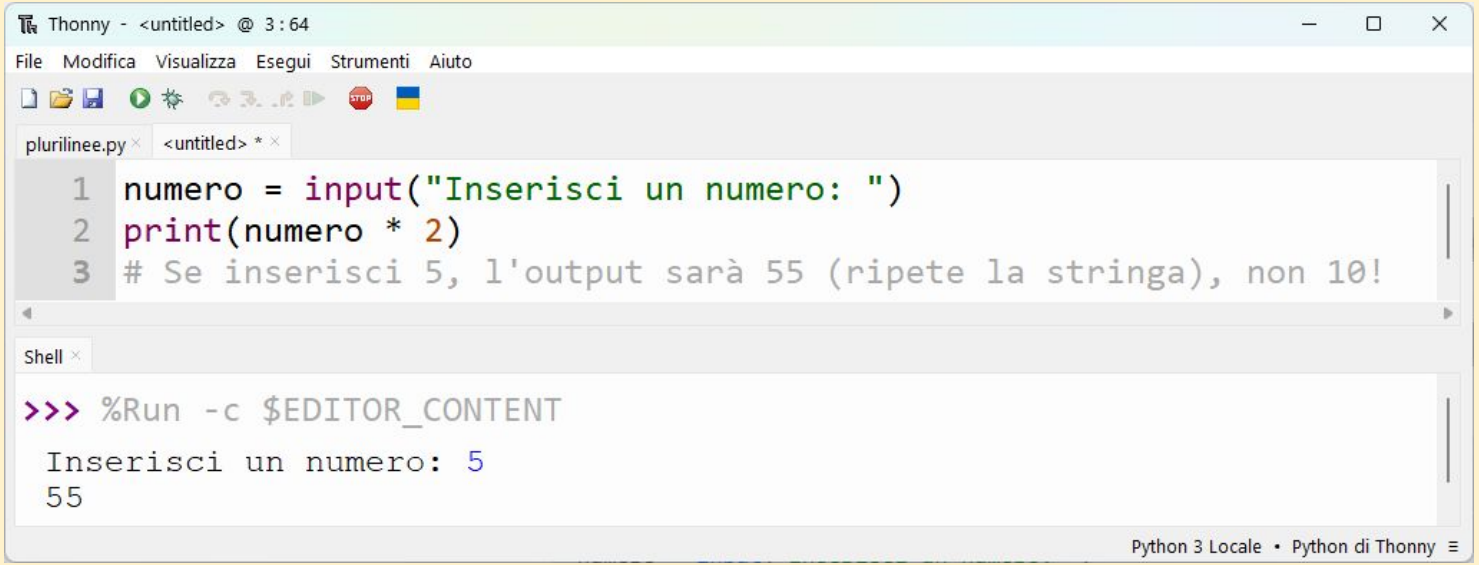
La funzione base: `input()`

La funzione `input()` interrompe l'esecuzione del programma e attende che l'utente scriva qualcosa e prema Invio.



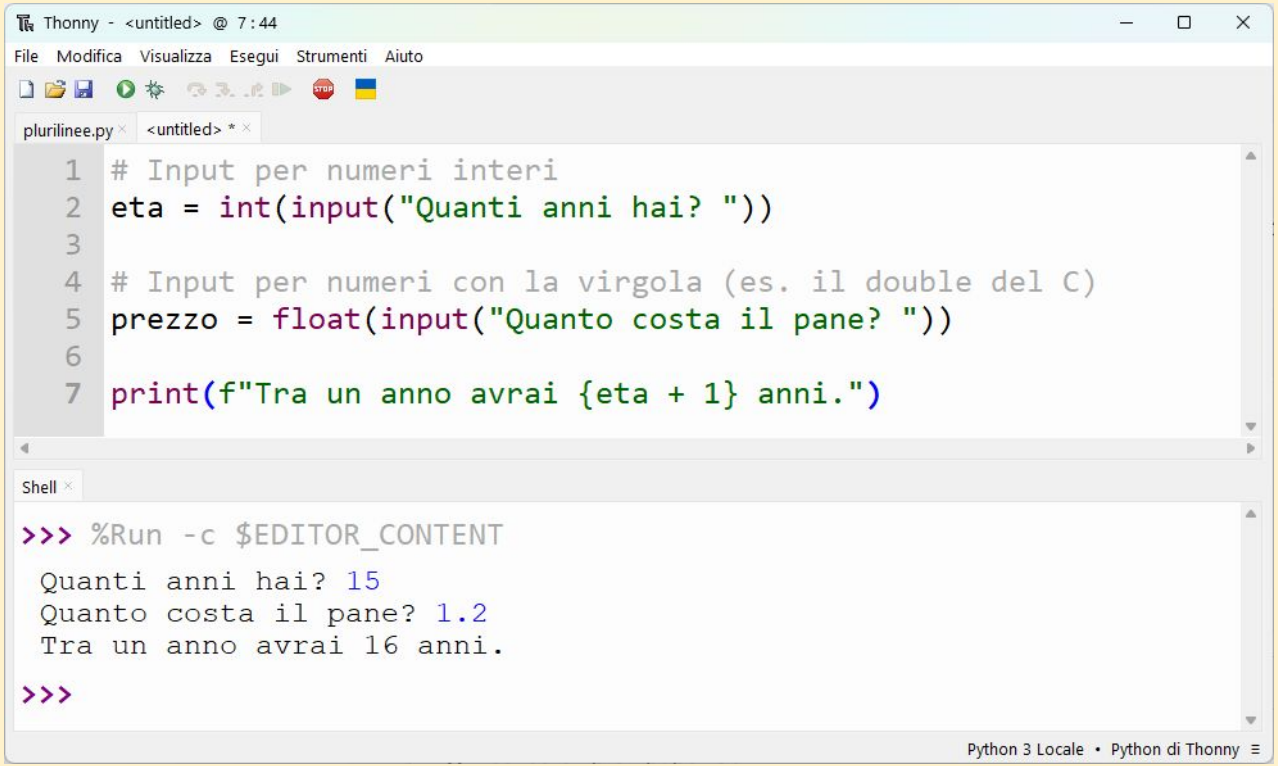
La regola d'oro: "È tutto testo"

Questa è la parte più importante da insegnare: **`input()` restituisce sempre una stringa (testo)**, anche se l'utente digita un numero. Se provi a fare calcoli su un input senza convertirlo, Python restituirà un errore o un risultato inaspettato:



La conversione (Casting)

Per usare l'input come numero, devi "avvolgere" la funzione `input()` dentro `int()` (per numeri interi) o `float()` (per numeri con la virgola).



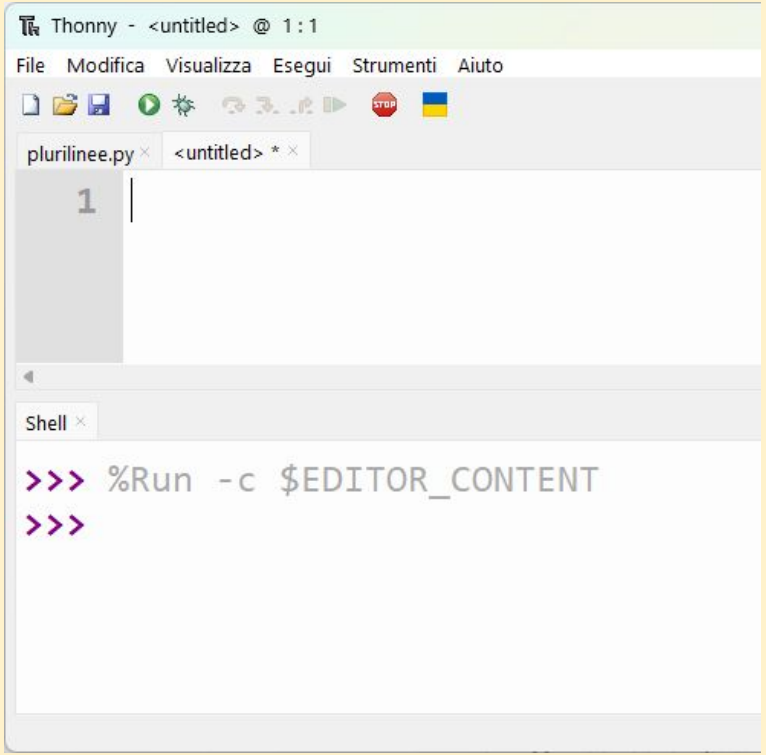
Schema riassuntivo del flusso di Input

Comando	Cosa ottieni	Uso tipico
input()	**Stringa** (str)	Nomi, parole, messaggi
int(input())	**Intero** (int)	Conteggi, età, anni
float(input())	**Decimale** (float)	Prezzi, misure, medie

Esercizio microscopico per gli studenti

Scrivere un programma di due righe che:

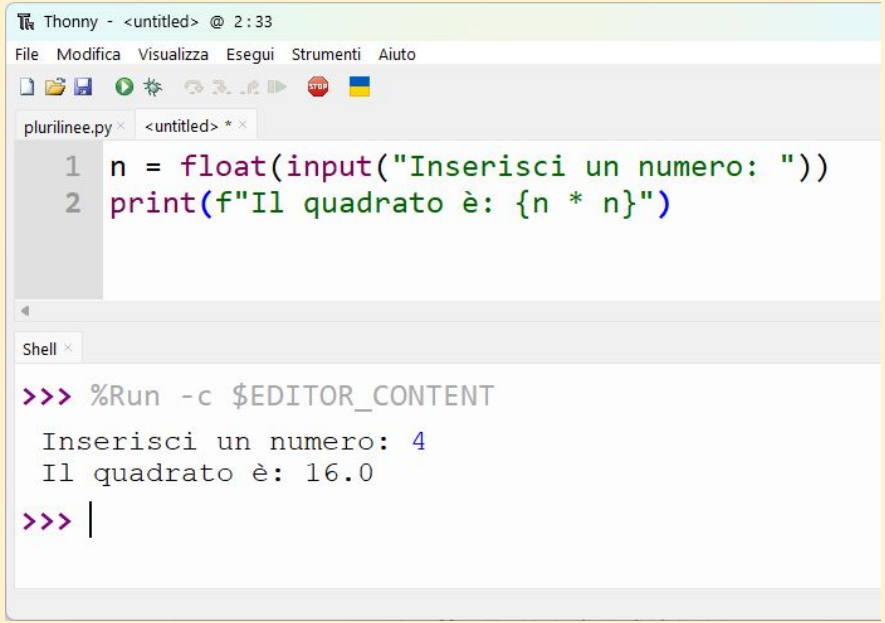
1. Chiede un numero all'utente.
2. Stampa il quadrato di quel numero.



Esercizio microscopico per gli studenti

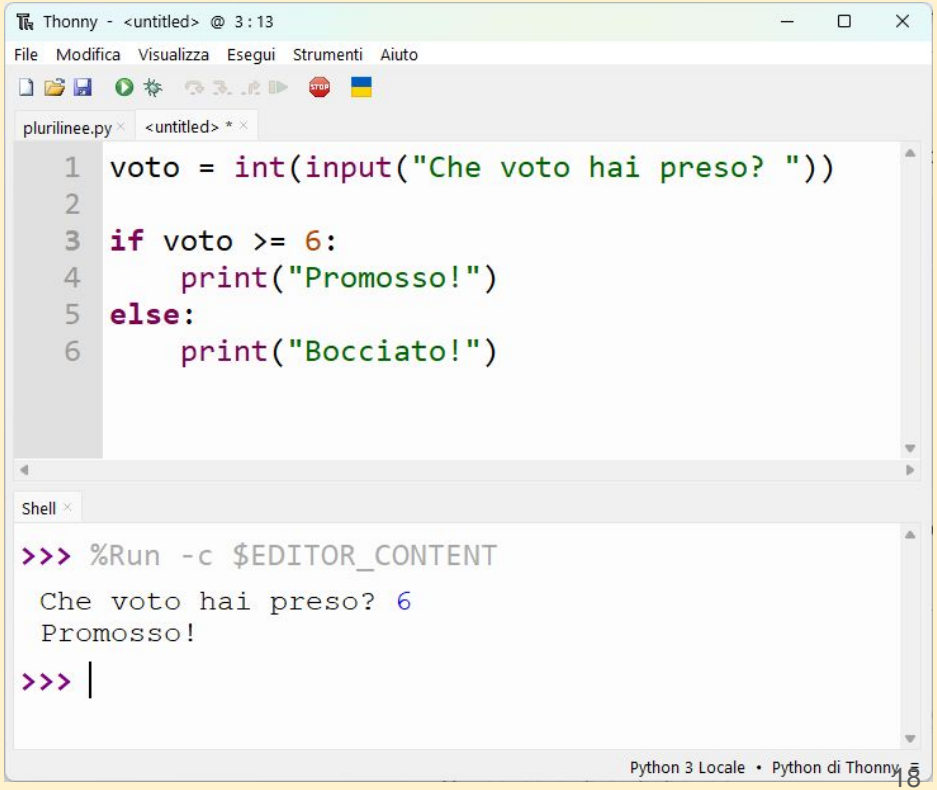
Scrivere un programma di due righe che:

1. Chiede un numero all'utente.
2. Stampa il quadrato di quel numero.



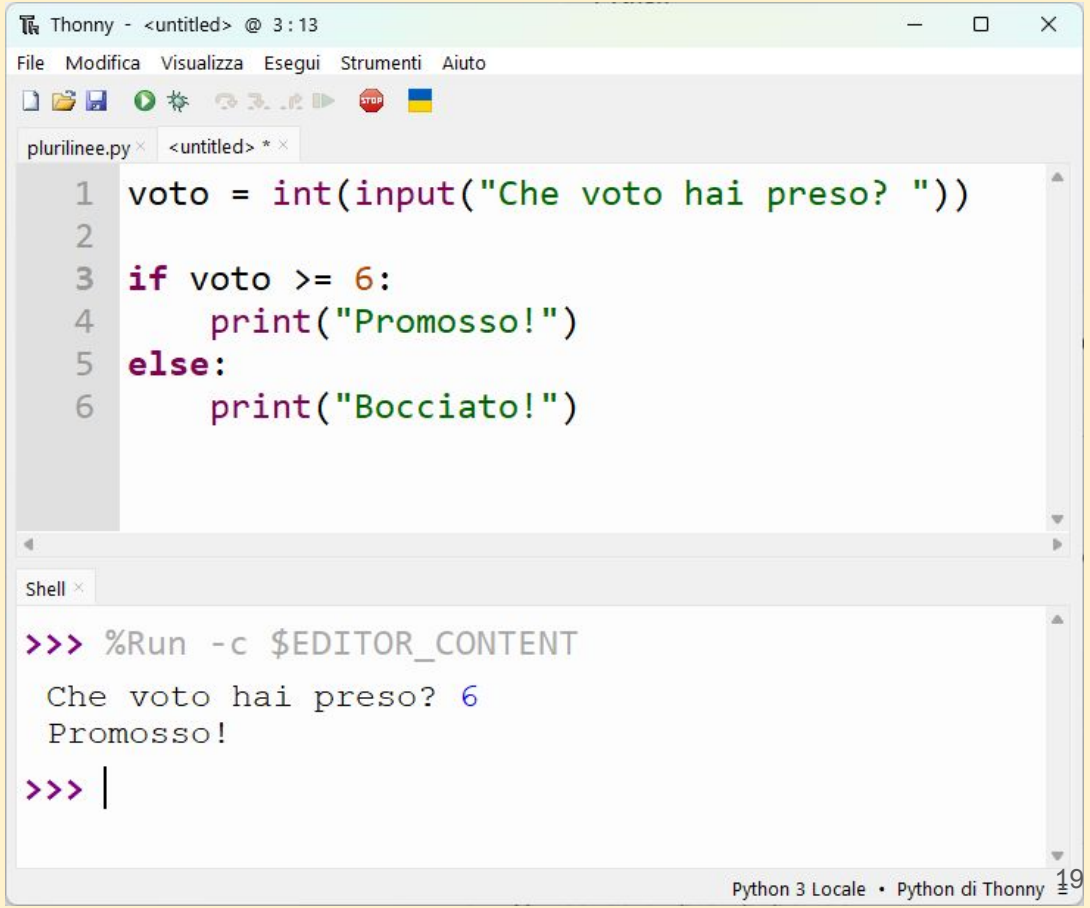
L' `if` in Python (Semplice e visivo)

L'istruzione **`if`** è il "centro decisionale" del codice. Permette al programma di eseguire determinate azioni **solo se** una condizione è vera. Se la condizione è falsa, il programma salta quella parte di codice.



Regola

Dopo l' `if` servono sempre i due punti (`:`) e il codice interno deve essere spostato a destra con il tasto Tab.

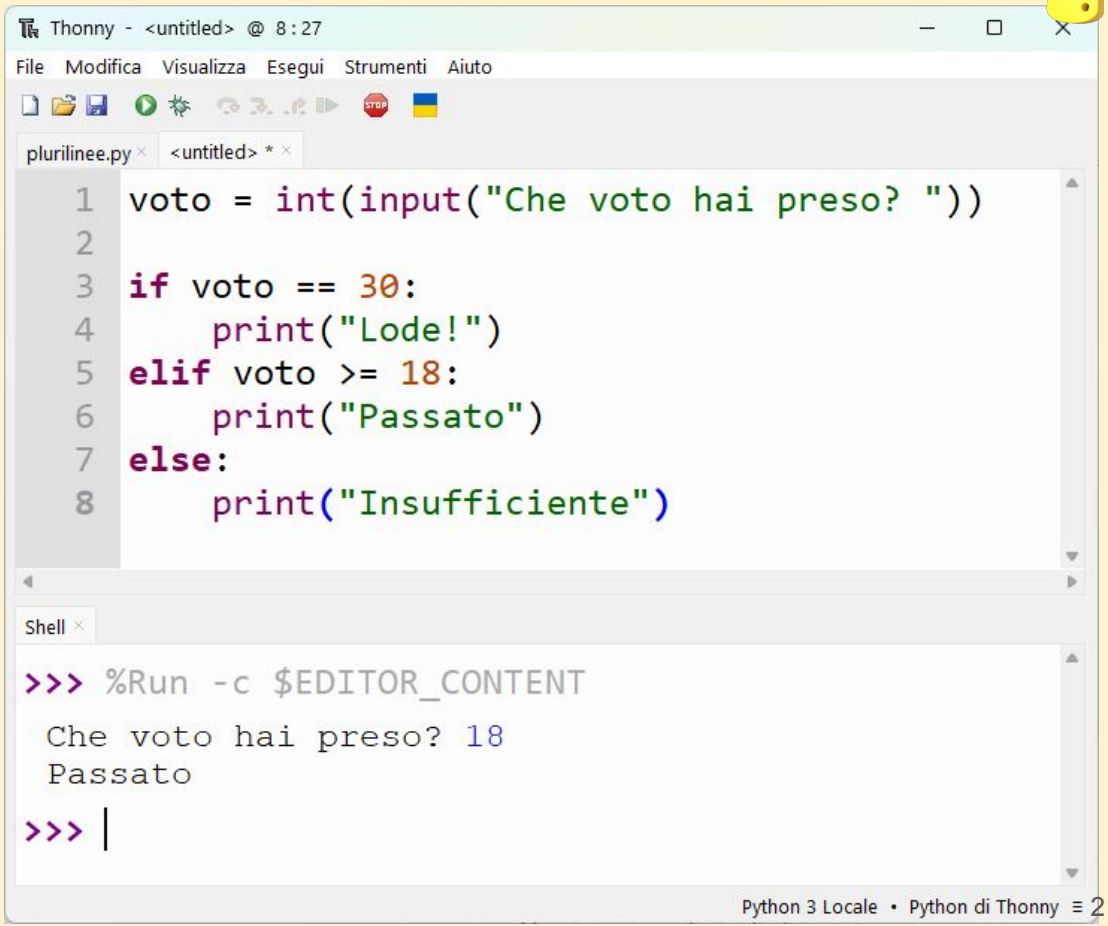


Gli Operatori di Confronto

Simbolo	Significato
`==`	Uguale a (attenzione: due uguali!)
`!=`	Diverso da
`>` / `<`	Maggiore / Minore
`>=` / `<=`	Maggiore o uguale / Minore o uguale

La variante "else if" (Scelte multiple)

Cosa succede se hai più di due opzioni? Usi una catena di controlli.



In C (`else if`):

```c
if (voto == 30) {
    printf("Lode!\n");
} else if (voto >= 18) {
    printf("Passato\n");
} else {
    printf("Insufficiente\n");
}
```

Nota per gli studenti: l'errore dell' `=`

L'errore più comune in assoluto è scrivere `if (voto = 18)`.

- **`=` (singolo):** Assegna un valore (metti 18 dentro la scatola voto).
- **`==` (doppio):** Confronta due valori (chiedi se nella scatola c'è il 18).

Traccia 1: Il Controllo Parità

Obiettivo: Chiedere un numero intero e dire se è **Pari** o **Dispari**. *Logica:* Un numero è pari se il resto della divisione per 2 è zero (operatore %).

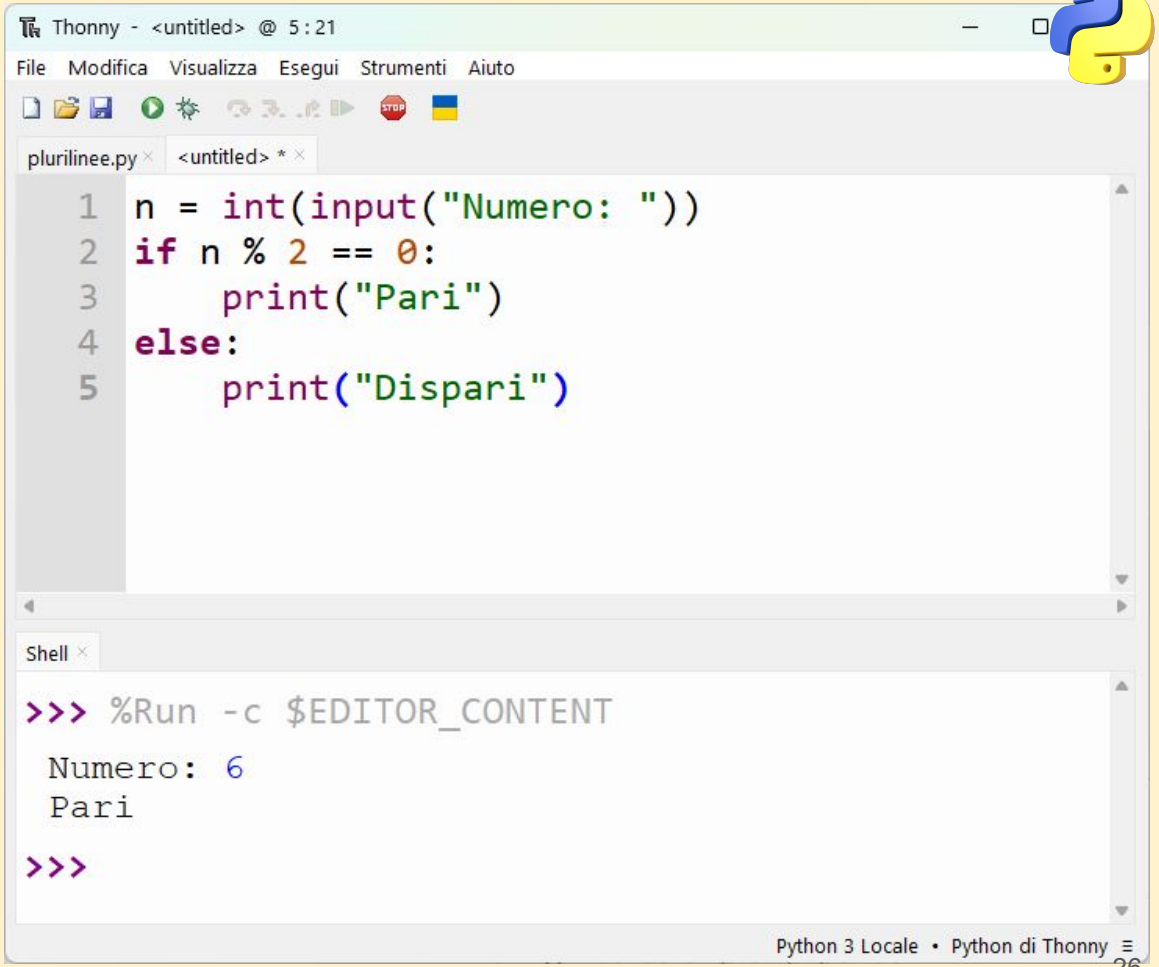
Linguaggio C

main.c

```c
#include <stdio.h>

int main() {
  int n;
  printf("Numero: ");
  scanf("%d", &n);
  if (n % 2 == 0) printf("Pari");
  else printf("Dispari");
}
```

```
Numero: 6
Pari

...Program finished with exit code 0
Press ENTER to exit console.
```

Linguaggio Python



Traccia 2: Lo Sconto "Precisione"

Obiettivo: Chiedere il prezzo di un prodotto (con decimali) e applicare uno sconto del 10% solo se il prezzo supera i 50 Euro. *Logica:* Usare tipi a virgola mobile e una condizione di soglia.

C

```c
double prezzo;
printf("Prezzo: ");
scanf("%lf", &prezzo);
if (prezzo > 50) prezzo = prezzo * 0.9;
printf("Totale: %.2lf", prezzo);
```

In Python:

Python

```python
prezzo = float(input("Prezzo: "))
if prezzo > 50:
    prezzo *= 0.9
print(f"Totale: {prezzo:.2f}")
```

Traccia 3: Il Termometro

Obiettivo: Chiedere la temperatura e stampare un messaggio:

- "Ghiaccio" se <=0,
- "Acqua" se positiva.

Logica: Gestione dei numeri negativi e confronto semplice.

In C:

C

```c
float temp;
printf("Temp: ");
scanf("%f", &temp);
if (temp <= 0) printf("Stato: Ghiaccio");
else printf("Stato: Liquido");
```

In Python:

Python

```python
temp = float(input("Temp: "))
if temp <= 0:
    print("Stato: Ghiaccio")
else:
    print("Stato: Liquido")
```

In Python, il ciclo `for` abbinato alla funzione `range()` è lo strumento standard per ripetere un'azione un numero preciso di volte. A differenza del C, dove devi gestire manualmente contatori e incrementi, Python si occupa di tutto "sotto il cofano".

La funzione `range()`

La funzione `range()` genera una sequenza di numeri. Può essere usata in tre modi diversi:

- **`range(n)`**: Parte da **0** e arriva a **n-1** (totale `n` ripetizioni).
- **`range(start, stop)`**: Parte da `start` e arriva a `stop - 1`.
- **`range(start, stop, step)`**: Come sopra, ma salta ogni volta di un valore `step`.

2. Esempi Microscopici

A. Ripetere 5 volte (da 0 a 4)

Python

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

B. Contare da 10 a 20

Python

```python
for i in range(10, 21):
    print(i) # Stampa i numeri da 10 a 20 inclusi
```

C. Solo numeri pari (usando lo step)

Python

```python
for i in range(0, 11, 2):
    print(i) # Stampa: 0, 2, 4, 6, 8, 10
```

Confronto con il linguaggio C

Per far capire la potenza di Python si dimostra come la stessa operazione sia molto più verbosa in C:

Caratteristica	Linguaggio C	Python
Sintassi	`for(int i=0; i<5; i++)`	`for i in range(5):`
Gestione	Devi dichiarare, testare e incrementare.	Python itera automaticamente sulla sequenza.
Leggibilità	Più tecnica e complessa.	Molto simile all'inglese parlato.

Due errori comuni da segnalare:

1. **L'ultimo numero è escluso:** `range(5)` non include il 5. È fondamentale ricordarlo quando si lavora con gli indici.
2. **I due punti e l'indentazione:** Come per l' `if`, il `for` richiede i `:` alla fine della riga e il codice interno deve essere spostato a destra (Tab).

Esercizi microscopici:

1. **Tabellina:** Stampa la tabellina del 5 (da 5 a 50) usando un `range` con lo step.
2. **Conto alla rovescia:** Usa `range(10, 0, -1)` per stampare un countdown da 10 a 1.
3. **Sommatoria:** Chiedi un numero `n` e usa un ciclo per sommare tutti i numeri da 1 a `n`.

Python

```python
for i in range(5, 51, 5):
    print(i)
```

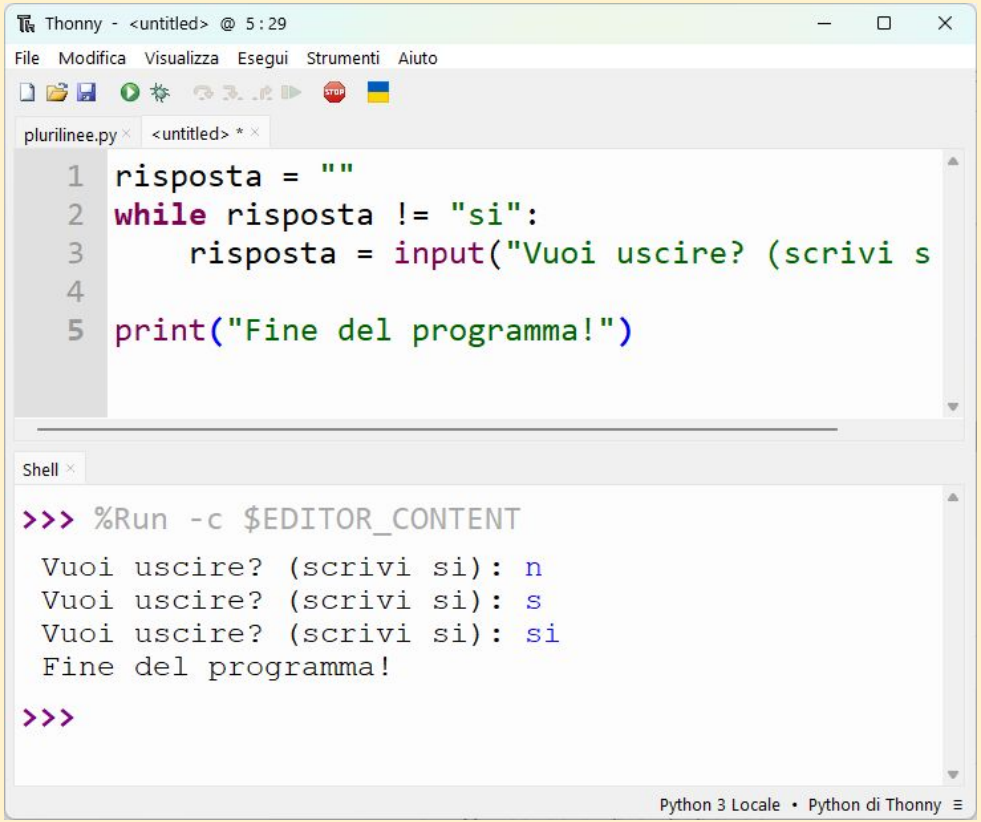
Python

```python
for i in range(10, 0, -1):
    print(i)
print("Decollo! 🚀")
```

Python

```python
n = int(input("Fino a che numero vuoi sommare? "))
somma = 0

for i in range(1, n + 1):
    somma = somma + i

print(f"La somma totale è: {somma}")
```

Mentre il `for` si usa quando sappiamo **quante volte** ripetere un'azione (es. "falla 10 volte"), il `while` si usa quando la ripetizione dipende da una **condizione** che può cambiare (es. "continua finché non premi il tasto ESC").



Esercizi microscopici sul `while`:

1. **Password:** Chiedi una password finché l'utente non inserisce quella corretta (es. "1234").
2. **Somma infinita:** Chiedi numeri all'utente e sommali tra loro. Fermati solo quando l'utente inserisce lo `0`.
3. **Conto alla rovescia personalizzato:** Chiedi un numero di partenza e scendi fino a 0 usando il `while`.

Ti servono le soluzioni di questi tre esercizi in Python?

1. Controllo Password

Questo esercizio è utilissimo per spiegare il concetto di **sicurezza** e iterazione infinita finché una condizione non è soddisfatta.

Python

```python
password_corretta = "1234"
tentativo = ""

while tentativo != password_corretta:
    tentativo = input("Inserisci la password: ")

print("Accesso consentito!")
```

2. Somma con "Sentinella"

In informatica, lo `0` in questo caso viene chiamato **valore sentinella**: serve a interrompere il ciclo.

Python

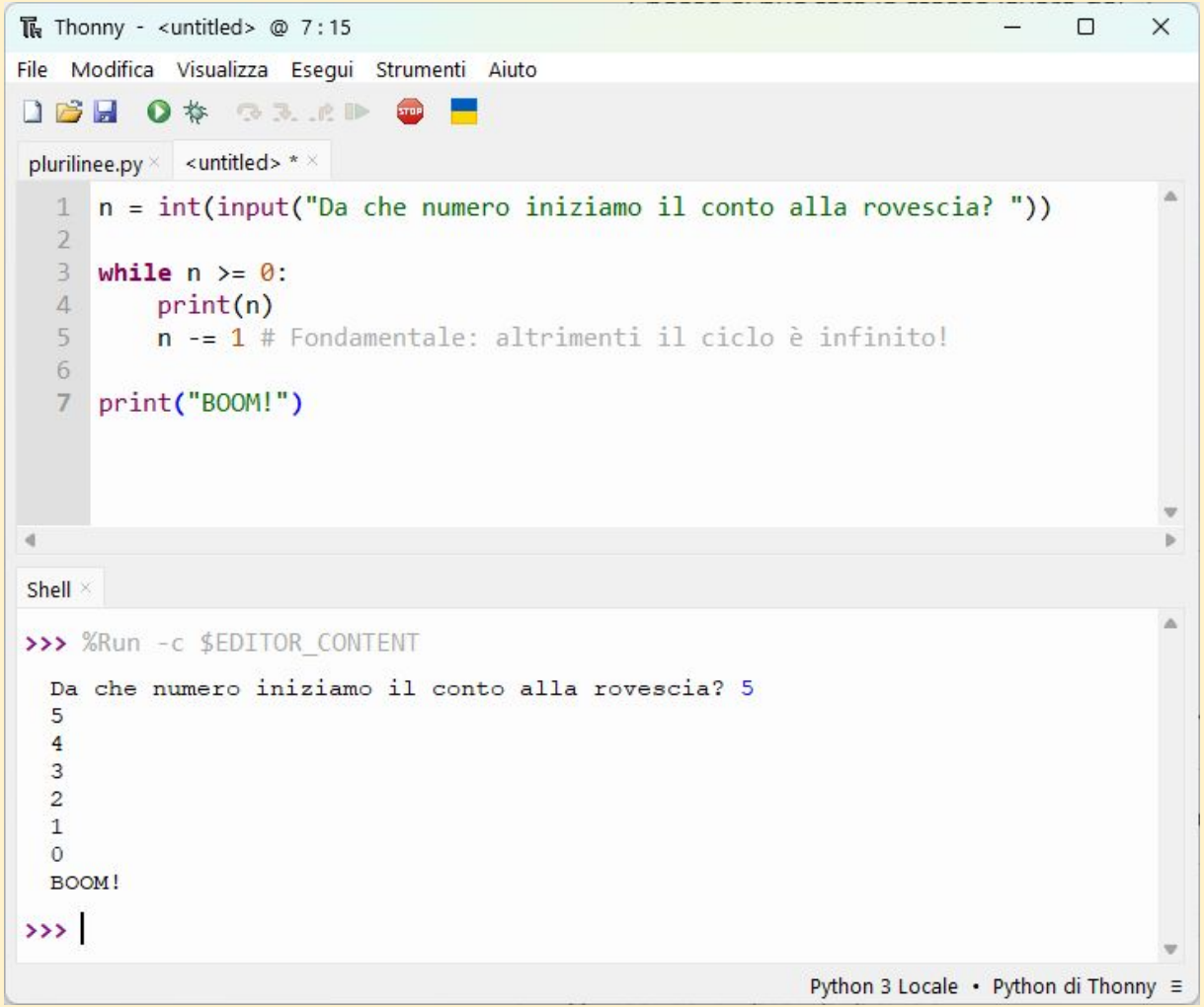
```python
totale = 0
numero = -1 # Inizializzo con un valore diverso da 0 per entrare nel ciclo

while numero != 0:
    numero = int(input("Inserisci un numero da sommare (0 per finire): "))
    totale += numero

print(f"Il totale finale è: {totale}")
```

3. Countdown con While

Spesso si può fare lo stesso lavoro del `for` con un `while`, ma bisogna gestire l'incremento (o decremento) a mano.

Python

```python
n = int(input("Da che numero iniziamo il conto alla rovescia? "))

while n >= 0:
    print(n)
    n -= 1 # Fondamentale: altrimenti il ciclo è infinito!

print("BOOM!")
```


Thonny - <untitled> @ 7 : 15
File Modifica Visualizza Esegui Strumenti Aiuto
plurilinee.py
<untitled> *
1 n = int(input("Da che numero iniziamo il conto alla rovescia? "))
2
3 while n >= 0:
4 print(n)
5 n -= 1 # Fondamentale: altrimenti il ciclo è infinito!
6
7 print("BOOM!")
Shell
>>> %Run -c $EDITOR_CONTENT
Da che numero iniziamo il conto alla rovescia? 5
5
4
3
2
1
0
BOOM!
>>>
Python 3 Locale • Python di Thonny

Le **liste** sono le "super-variabili" di Python. Mentre una variabile normale può contenere un solo dato (es. `x = 5`), una lista può contenere un'intera collezione di oggetti, ordinati e modificabili.

```
frutta = ["Mela", "Banana", "Ciliegia"]
numeri = [10, 20, 30, 40]
mista = ["Ciao", 10, 3.14] # Python permette anche tipi diversi!
```

Accedere agli elementi (L'indice)

Per prendere un elemento, si usa la sua posizione (indice). **Attenzione:** in informatica si inizia sempre a contare da **0**.

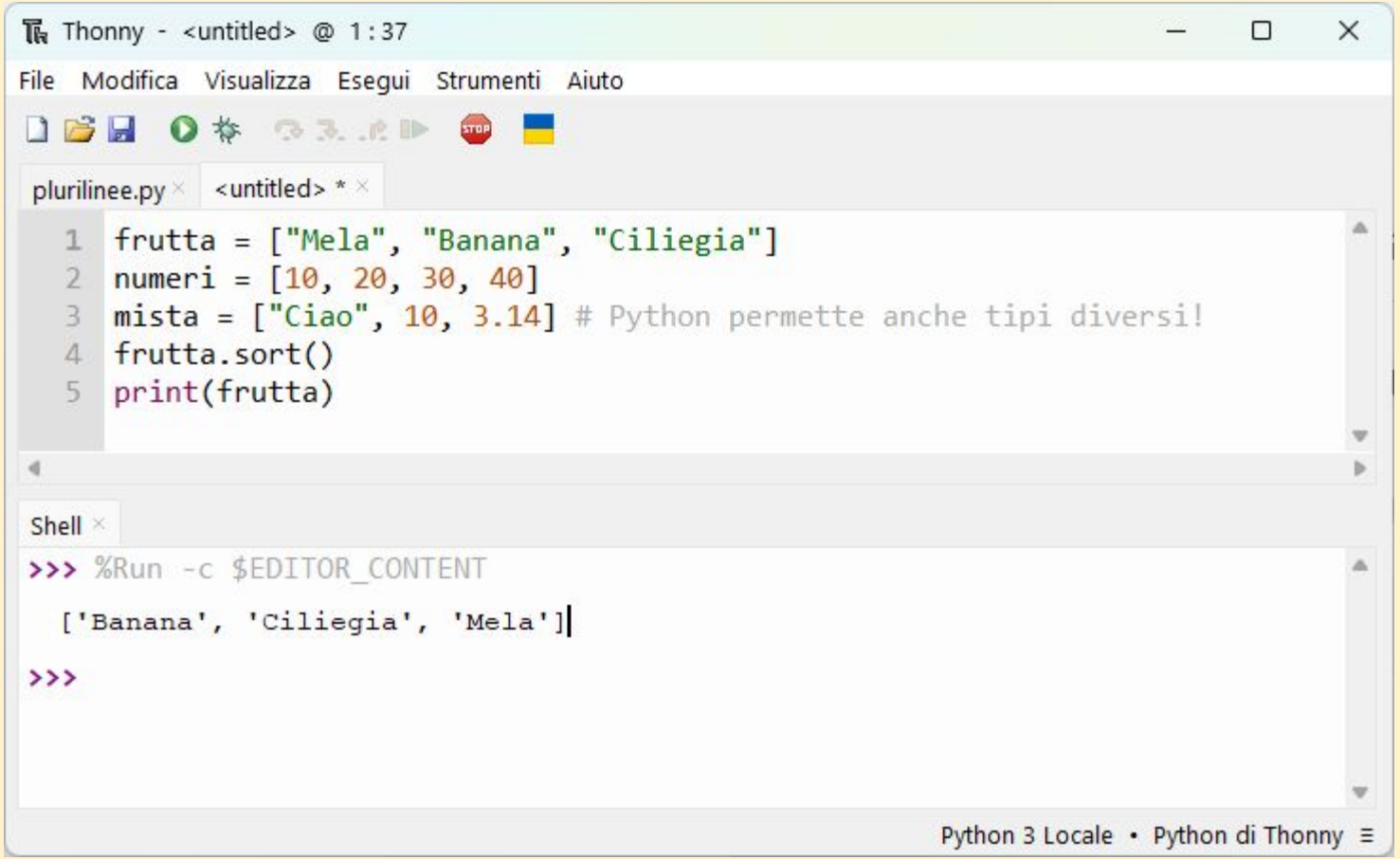
```
colori = ["Rosso", "Verde", "Blu"]

print(colori[0]) # Stampa: Rosso
print(colori[1]) # Stampa: Verde
```

Operazioni comuni (Metodi)

Le liste hanno dei "superpoteri" incorporati per essere modificate:

- **`append(valore)`**: Aggiunge un elemento alla fine.
- **`len(lista)`**: Ti dice quanti elementi ci sono (la lunghezza).
- **`pop()`**: Rimuove l'ultimo elemento.
- **`sort()`**: Ordina la lista (es. in ordine alfabetico o numerico).

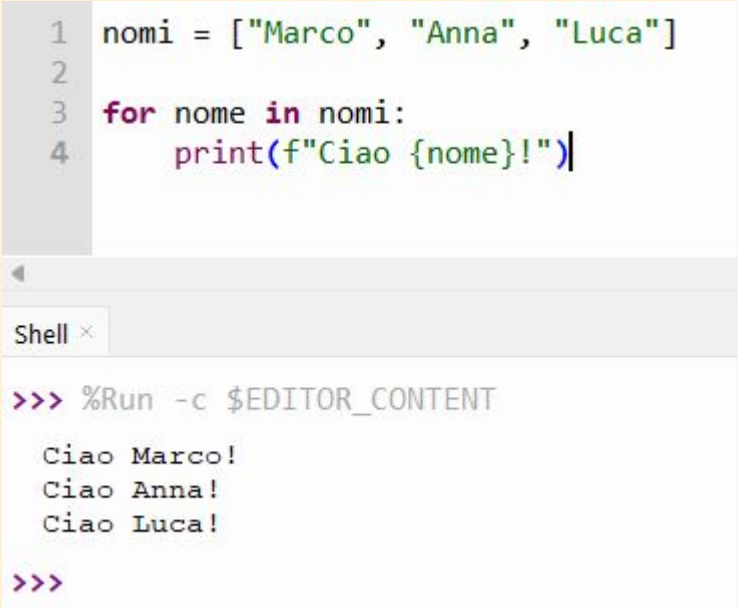
```
spesa = ["Pane"]
spesa.append("Latte") # La lista ora è ["Pane", "Latte"]
print(len(spesa))     # Stampa: 2
```


Thonny - <untitled> @ 1:37
File Modifica Visualizza Esegui Strumenti Aiuto
plurilinee.py
<untitled> *
1 frutta = ["Mela", "Banana", "Ciliegia"]
2 numeri = [10, 20, 30, 40]
3 mista = ["Ciao", 10, 3.14] # Python permette anche tipi diversi!
4 frutta.sort()
5 print(frutta)
Shell
>>> %Run -c $EDITOR_CONTENT
['Banana', 'Ciliegia', 'Mela']
>>>
Python 3 Locale • Python di Thonny

Ciclo FOR + Liste

Il modo più comune per usare le liste è scorrerle con un ciclo `for`. In Python è quasi come leggere una frase

```
nomi = ["Marco", "Anna", "Luca"]

for nome in nomi:
    print(f"Ciao {nome}!")
```



Esercizi microscopici sulle liste:

1. **La lista dei desideri:** Crea una lista vuota `regali = []`. Usa `append()` per aggiungere tre oggetti chiesti all'utente con un `input`.
2. **Il voto più alto:** Data una lista di numeri `voti = [24, 28, 18, 30, 22]`, usa la funzione `max(voti)` per stampare il voto più alto e `len(voti)` per dire quanti esami sono stati fatti.
3. **Cambio elemento:** Crea una lista di 3 colori. Chiedi all'utente un nuovo colore e usalo per sostituire il colore alla posizione `0`.

1. La lista dei desideri (Aggiunta dinamica)

In questo esercizio usiamo un ciclo `for` per non ripetere tre volte lo stesso codice. È il modo più pulito di procedere.

Python

```python
regali = []

for i in range(3):
    oggetto = input(f"Cosa vorresti per regalo? ({i+1}/3): ")
    regali.append(oggetto)

print(f"La tua lista dei desideri è: {regali}")
```

2. Il voto più alto (Analisi dei dati)

Qui sfruttiamo le funzioni integrate di Python (`max` e `len`) che rendono superfluo scrivere algoritmi complessi per trovare il valore maggiore.

Python

```python
voti = [24, 28, 18, 30, 22]

voto_massimo = max(voti)
numero_esami = len(voti)

print(f"Il voto più alto è: {voto_massimo}")
print(f"In totale hai sostenuto {numero_esami} esami.")
```

3. Cambio elemento (Sostituzione tramite indice)

Questo esercizio serve a far capire che le liste sono **mutabili**: puoi cambiare un pezzetto della lista senza doverla ricreare da zero.

Python

```python
colori = ["Rosso", "Verde", "Blu"]
print(f"Colori attuali: {colori}")

nuovo_colore = input("Inserisci un nuovo colore per sostituire il primo: ")

# Sostituiamo l'elemento in posizione 0
colori[0] = nuovo_colore

print(f"Lista aggiornata: {colori}")
```

Riepilogo per la classe

Operazione	Codice	Effetto
Aggiungere	lista.append(x)	Mette x in fondo alla lista.
Contare	len(lista)	Restituisce il numero di elementi.
Trovare il max	max(lista)	Trova il valore più grande (numerico).
Sostituire	lista[0] = x	Sovrascrive l'elemento alla posizione indicata.

Le **funzioni** sono blocchi di codice isolati che hanno un nome e che possono essere "chiamati" ogni volta che servono.
Servono a due scopi fondamentali:

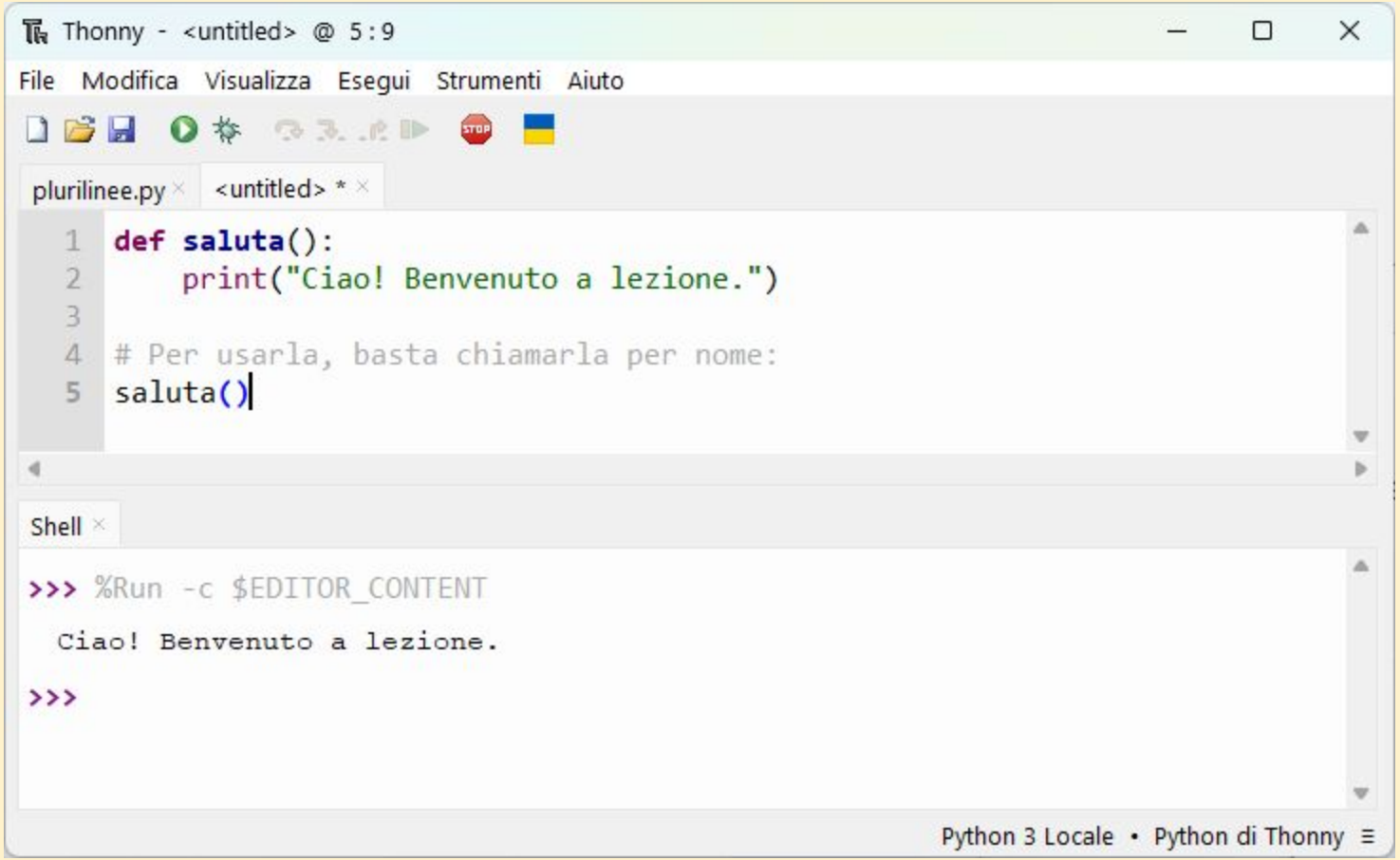
1. **Evitare ripetizioni:** Non scrivi 10 volte lo stesso codice, ma chiami la funzione.
2. **Ordine:** Dividono un programma gigante in piccoli pezzi facili da gestire.

1. La sintassi in Python

In Python si usa la parola chiave `def` (che sta per *define*).

Python

```python
def saluta():
    print("Ciao! Benvenuto a lezione.")

# Per usarla, basta chiamarla per nome:
saluta()
```

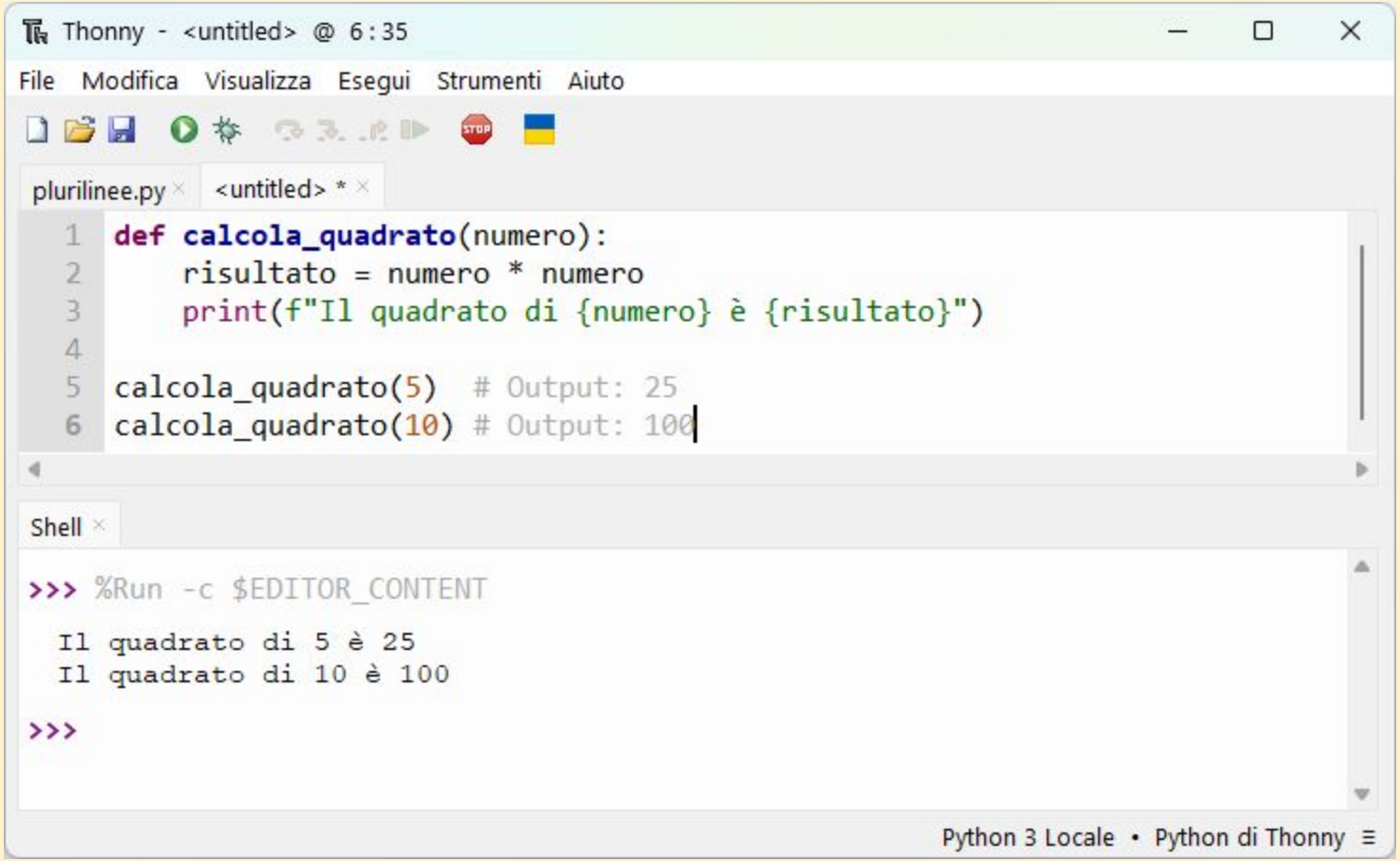

Thonny - <untitled> @ 5:9
File Modifica Visualizza Esegui Strumenti Aiuto
plurilinee.py
<untitled> *
1 def saluta():
2 print("Ciao! Benvenuto a lezione.")
3
4 # Per usarla, basta chiamarla per nome:
5 saluta()
Shell
>>> %Run -c $EDITOR_CONTENT
Ciao! Benvenuto a lezione.
>>>
Python 3 Locale • Python di Thonny

2. Parametri (Dati in ingresso)

Le funzioni diventano utili quando possono elaborare dati diversi. Questi dati si chiamano **parametri**.

Python

```python
def calcola_quadrato(numero):
    risultato = numero * numero
    print(f"Il quadrato di {numero} è {risultato}")

calcola_quadrato(5)  # Output: 25
calcola_quadrato(10) # Output: 100
```


Thonny - <untitled> @ 6 : 35
File Modifica Visualizza Esegui Strumenti Aiuto
plurilinee.py
<untitled> *
1 def calcola_quadrato(numero):
2 risultato = numero * numero
3 print(f"Il quadrato di {numero} è {risultato}")
4
5 calcola_quadrato(5) # Output: 25
6 calcola_quadrato(10) # Output: 100
Shell
>>> %Run -c $EDITOR_CONTENT
Il quadrato di 5 è 25
Il quadrato di 10 è 100
>>>
Python 3 Locale • Python di Thonny

3. Il valore di ritorno (`return`)

A volte non vogliamo che la funzione stampi qualcosa, ma che ci "restituisca" un risultato da usare in un altro calcolo.

Python

```python
def somma(a, b):
    return a + b

risultato_finale = somma(10, 5) # La funzione "diventa" 15
print(risultato_finale * 2)     # Output: 30
```

3 Esercizi microscopici sulle funzioni:

1. **Funzione "Area":** Crea una funzione chiamata `area_rettangolo(base, altezza)` che restituisce il prodotto dei due valori.
2. **Funzione "Check Voto":** Crea una funzione `esito(voto)` che stampa "Passato" se il voto è ≥ 18, altrimenti stampa "Riprova".
3. **Funzione "Media Lista":** Crea una funzione che riceve una lista di numeri e restituisce la loro media (usa `sum(lista) / len(lista)`).

1. Funzione "Area" (Uso del `return`)

Questa funzione calcola un valore e lo "consegna" al programma principale.

Python

```python
def area_rettangolo(base, altezza):
    return base * altezza

# Test della funzione
risultato = area_rettangolo(5, 10)
print(f"L'area del rettangolo è: {risultato}")
```

2. Funzione "Check Voto" (Logica interna)

Questa funzione non restituisce nulla, ma esegue un'azione (la stampa) in base a una condizione.

Python

```python
def esito(voto):
    if voto >= 18:
        print("Esito: Passato")
    else:
        print("Esito: Riprova")

# Test della funzione
voto_studente = int(input("Inserisci il voto: "))
esito(voto_studente)
```

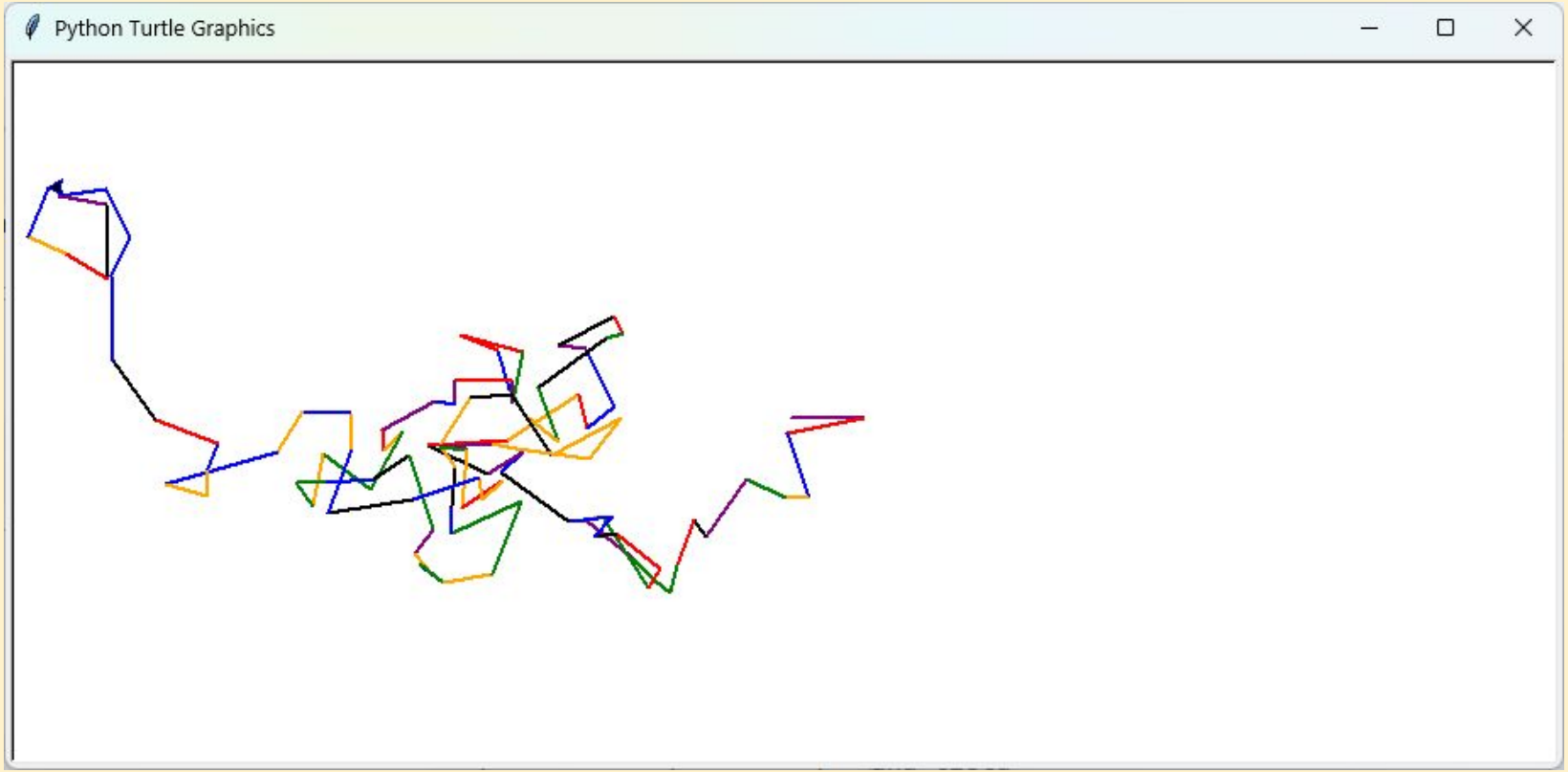
3. Funzione "Media Lista" (Funzioni che accettano liste)

In Python puoi passare un'intera lista come singolo parametro. È un modo molto potente per elaborare gruppi di dati.

Python

```python
def calcola_media(numeri):
    if len(numeri) == 0:
        return 0
    return sum(numeri) / len(numeri)

# Test della funzione
miei_voti = [24, 30, 18, 22]
media = calcola_media(miei_voti)
print(f"La tua media è: {media:.2f}")
```

Turtle



from turtle import *

Con questo comando entri nel magico mondo della **Turtle Graphics**, uno dei modi più divertenti e visivi per insegnare la logica della programmazione (e la geometria) agli studenti.

In pratica, controlli una "piccola tartaruga" che si muove su un piano cartesiano trascinando una penna e lasciando una traccia.

`import *` significa "importa tutto". È comodo per i principianti perché scrive meno codice, ma nei programmi grandi si preferisce `import turtle` per evitare confusione tra i nomi delle funzioni.

I comandi base (Il vocabolario della tartaruga)

Quando usi `from turtle import *`, non hai bisogno di scrivere `turtle.forward()`, ma puoi usare direttamente i comandi:

- **`forward(distanza)`** o **`fd()`**: Va avanti.
- **`backward(distanza)`** o **`bk()`**: Va indietro.
- **`right(angolazione)`** o **`rt()`**: Ruota a destra di un tot di gradi.
- **`left(angolazione)`** o **`lt()`**: Ruota a sinistra.
- **`pencolor("color")`**: Cambia il colore della penna (es. "red", "blue").
- **`pensize(spessore)`**: Cambia lo spessore della traccia.

2. Esempio: Disegnare un Quadrato

Qui puoi mostrare come combinare il ciclo `for` che abbiamo visto prima con i movimenti della tartaruga.

Python

```
from turtle import *

shape("turtle") # Trasforma il cursore in una tartaruga
speed(1)        # Imposta una velocità lenta per far vedere i passaggi

for i in range(4):
    forward(100)
    left(90)

done() # Mantiene la finestra aperta alla fine del disegno
```

Alzare e abbassare la penna

Fondamentale per spostarsi senza lasciare segni:

- `penup()` o `up()`: Solleva la penna (si muove senza scrivere).
- `pendown()` o `down()`: Abbassa la penna (ricomincia a scrivere).

Esercizi microscopici con Turtle:

1. **Triangolo Equilatero:** Disegna un triangolo. (Suggerimento per gli studenti: l'angolo esterno da girare è di **120°**, non 60!).
2. **Cerchio Colorato:** Usa color("green") e poi il comando circle(50).
3. **Il "Pentagono":** Usa un ciclo for per creare un poligono di 5 lati (l'angolo di rotazione è $360 / 5 = 72$).

1. Triangolo Equilatero

Per chiudere un poligono regolare, la somma degli angoli esterni deve essere sempre **360°**.

Quindi, per un triangolo: $360/3 = 120$.

Python

```python
from turtle import *

shape("turtle")
color("red")

for i in range(3):
    forward(100)
    left(120)  # L'angolo esterno per girare correttamente

done()
```

2. Cerchio Colorato

Python ha un comando dedicato per i cerchi. È molto utile per spiegare il concetto di parametri: il numero dentro `circle()` è il **raggio**.

Python

```
from turtle import *

shape("turtle")
pensize(5)      # Rende il tratto più spesso
pencolor("green")

circle(50)      # Disegna un cerchio con raggio 50

done()
```

3. Il Pentagono

Qui gli studenti devono fare un piccolo calcolo logico: $360/5 = 72$ gradi di rotazione.

Python

```python
from turtle import *

shape("turtle")
color("blue")

for i in range(5):
    forward(80)
    left(72)

done()
```

Bonus: La "Spirale Magica" (Per stupire la classe)

Puoi mostrare come un piccolo cambiamento nel ciclo `for` crei figure complesse. Qui la tartaruga aumenta la distanza di un pochino a ogni giro e gira di un angolo leggermente diverso da 90°.

Python

```python
from turtle import *
speed(0) # Velocità massima
pencolor("purple")

for i in range(100):
    forward(i * 2) # La linea diventa sempre più lunga
    left(91)       # Ruota quasi come un quadrato, ma "sfasa"

done()
```

Figura	Lati	Angolo di rotazione (360/lati)
Triangolo	3	120°
Quadrato	4	90°
Pentagono	5	72°
Esagono	6	60°
Cerchio	infiniti	circle()

Random

Random

In Python, per generare numeri casuali si utilizza il modulo `random`. È uno strumento fondamentale per creare giochi, simulazioni o per scegliere elementi a sorte da una lista.

Per usarlo, devi sempre scrivere `import random` all'inizio del file.

1. Generare numeri casuali

Ecco le tre funzioni più utilizzate dagli studenti:

- `random.randint(a, b)`: Genera un numero intero casuale compreso tra `a` e `b` (**inclusi**). È perfetto per simulare il lancio di un dado.
- `random.random()`: Genera un numero decimale (float) tra **0.0** e **1.0**.
- `random.uniform(a, b)`: Genera un numero decimale casuale tra `a` e `b`.

Random

Python

```python
import random

dado = random.randint(1, 6)
print(f"Hai lanciato un: {dado}")

percentuale = random.random()
print(f"Probabilità: {percentuale:.2%}")
```

Random

Scelte casuali da una lista

Se hai una lista di elementi (nomi, colori, oggetti), puoi estrarne uno o più a sorte:

- `random.choice(lista)`: Estrae un singolo elemento casuale.
- `random.shuffle(lista)`: Mescola gli elementi della lista originale (come un mazzo di carte).

Random

Python

```python
studenti = ["Luca", "Anna", "Marco", "Sara"]

# Estrazione del fortunato
estratto = random.choice(studenti)
print(f"L'interrogato è: {estratto}")

# Mescoliamo la classe
random.shuffle(studenti)
print(f"Ordine di presentazione: {studenti}")
```

Random

Esercizi microscopici con `random`:

1. **Indovina il numero:** Il computer sceglie un numero tra 1 e 10. Chiedi all'utente di indovinarlo con un `input` e un `if`.
2. **Lancio della moneta:** Crea un programma che generi casualmente "Testa" o "Croce".
3. **Il Disegno Caotico (con Turtle):** Usa un ciclo `for` da 100 giri. In ogni giro, la tartaruga deve girare di un angolo casuale tra 0 e 360 e andare avanti di una distanza casuale tra 10 e 50.

Random

1. Indovina il numero

Questo esercizio è perfetto per ripassare `input`, `if` e il casting a `int`.

Python

```python
import random

segreto = random.randint(1, 10)
scelta = int(input("Indovina il numero da 1 a 10: "))

if scelta == segreto:
    print("Incredibile! Hai indovinato! 🎉")
else:
    print(f"Peccato, il numero era {segreto}. Riprova!")
```

Random

2. Lancio della moneta

Qui puoi mostrare come usare `random.choice` per gestire valori testuali invece che numerici.

Python

```python
import random

opzioni = ["Testa", "Croce"]
risultato = random.choice(opzioni)

print(f"È uscito: {risultato}")
```

Soluzione della Sfida: I due Dadi

Ecco come combinare la somma e la condizione:

Python

```python
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
somma = dado1 + dado2

print(f"Dadi: {dado1} + {dado2} = {somma}")

if somma == 12:
    print("Vittoria! Hai fatto il punteggio massimo! 🏆")
else:
    print("Non hai vinto, ritenta.")
```

3. Il Disegno Caotico (Random + Turtle)

Questo esercizio entusiasma sempre gli studenti perché crea un'opera d'arte astratta diversa ogni volta che viene eseguito il codice.

Python

```python
from turtle import *
import random

speed(0) # Massima velocità
pensize(2)

for i in range(100):
    # Scegliamo un colore a caso da una lista
    colori = ["red", "blue", "green", "orange", "purple", "black"]
    pencolor(random.choice(colori))

    distanza = random.randint(10, 50)
    angolo = random.randint(0, 360)

    forward(distanza)
    left(angolo)

done()
```

Esercizi su Liste

1. Il Filtro dei Voti (Selezione)

Obiettivo: Data una lista di voti, creane una nuova che contenga solo i voti sufficienti (≥ 18).

Concetto: Scorrere una lista e popolare una seconda lista vuota.

Python

```python
voti_classe = [15, 22, 12, 30, 18, 25, 14]
sufficienti = []

for v in voti_classe:
    if v >= 18:
        sufficienti.append(v)

print(f"Voti totali: {voti_classe}")
print(f"Solo i sufficienti: {sufficienti}")
```

2. Il "Cerca Nomi" (Ricerca)

Obiettivo: Creare una lista di invitati. Chiedere all'utente un nome e dirgli se è nella lista oppure no.

Concetto: Uso dell'operatore `in`, che in Python è molto potente e leggibile.

Python

```python
invitati = ["Alice", "Bob", "Charlie", "Dora"]
nome_cercato = input("Chi stai cercando? ")

if nome_cercato in invitati:
    print(f"Sì, {nome_cercato} è presente alla festa! 🎉")
else:
    print(f"Mi dispiace, {nome_cercato} non è sulla lista. ⛔")
```

3. La Spesa Intelligente (Modifica)

Obiettivo: Data una lista della spesa, permetti all'utente di sostituire un elemento indicando l'indice.

Concetto: Gestione degli indici e consapevolezza che le liste partono da `0`.

Python

```python
spesa = ["Pane", "Latte", "Uova", "Pasta"]

print("Ecco la tua lista attuale:")
for i in range(len(spesa)):
    print(f"{i}: {spesa[i]}")

indice = int(input("\nQuale numero vuoi sostituire? "))
nuovo_prodotto = input("Cosa vuoi comprare invece? ")

spesa[indice] = nuovo_prodotto

print(f"Nuova lista aggiornata: {spesa}")
```

4. Massimo e Minimo "Manuali" (Algoritmi)

Obiettivo: Senza usare `max()` o `min()`, trova il numero più grande in una lista.

Perché farlo? Serve a far capire come ragiona il computer passo dopo passo.

Python

```python
numeri = [12, 45, 7, 89, 32]
piu_grande = numeri[0] # Ipotizzo che il primo sia il maggiore

for n in numeri:
    if n > piu_grande:
        piu_grande = n # Se ne trovo uno maggiore, lo sostituisco

print(f"Il numero più grande trovato è: {piu_grande}")
```

Obiettivo	Comando
Aggiungere in coda	lista.append("nuovo")
Rimuovere per nome	lista.remove("elemento")
Svuotare tutto	lista.clear()
Invertire l'ordine	lista.reverse()
Trovare la posizione	lista.index("elemento")

Fine