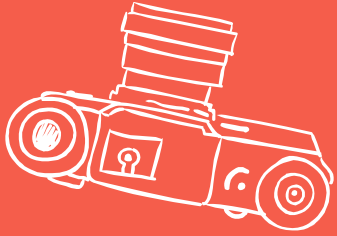


LABORATORIO DI PENSIERO COMPUTAZIONALE



Simone Zennaro – simone.zennaro@unipd.it



NELLA LEZIONE DI OGGI

- ✕ Esercizi
- ✕ Le variabili
- ✕ Le procedure e le funzioni



ESERCIZIO 4

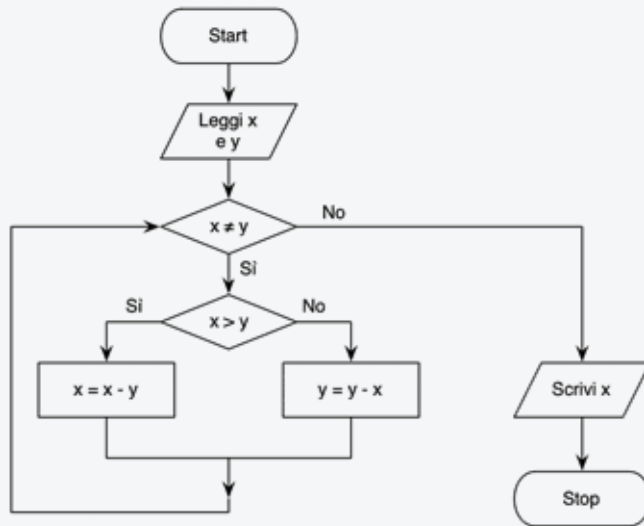
Dati due numeri interi positivi x e y calcolare il massimo comune divisore tra x e y usando somma e sottrazione.

ESERCIZIO 4



Dati due numeri interi positivi x e y calcolare il massimo comune divisore tra x e y .

Per costruire un algoritmo elementare per calcolare il MCD (massimo comun divisore) tra x e y ancora una volta facciamo ricorso ad una divisione piuttosto elementare in modo da costruire la strategia risolutiva basandoci soltanto su operazioni aritmetiche di base. In particolare l'idea è quella di applicare una variante del cosiddetto "algoritmo di Euclide": si tratta di sottrarre ripetutamente x da y (se $y > x$) o y da x (se $x > y$) fino ad ottenere lo stesso valore per x e y ; tale valore è proprio il massimo comun divisore dei due numeri originali.



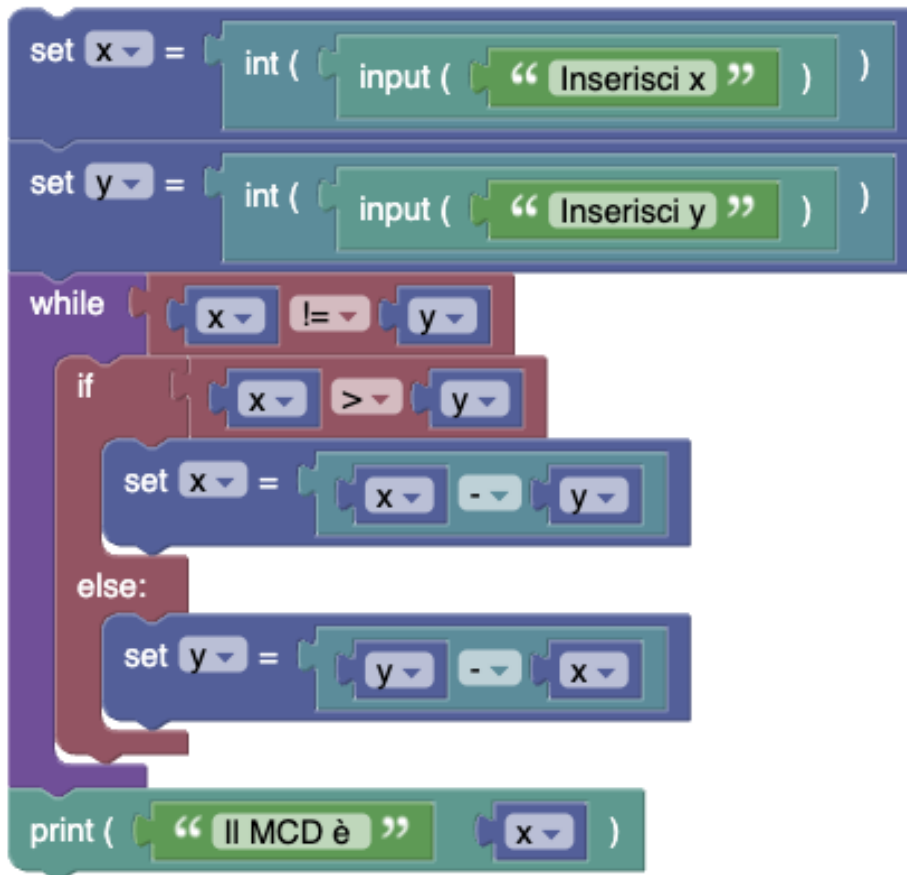
```
1: leggi x e y
3: fintanto che  $x \neq y$  ripeti
4:   se  $x > y$  allora
5:      $x = x - y$ 
6:   altrimenti
7:      $y = y - x$ 
8: scrivi "il MCD è x"
9: stop
```

ESERCIZIO 4



Dati due numeri interi positivi x e y calcolare il massimo comune divisore tra x e y .

Per costruire un algoritmo elementare per calcolare il MCD (massimo comun divisore) tra x e y ancora una volta facciamo ricorso ad una divisione piuttosto elementare in modo da costruire la strategia risolutiva basandoci soltanto su operazioni aritmetiche di base. In particolare l'idea è quella di applicare una variante del cosiddetto "algoritmo di Euclide": si tratta di sottrarre ripetutamente x da y (se $y > x$) o y da x (se $x > y$) fino ad ottenere lo stesso valore per x e y ; tale valore è proprio il massimo comun divisore dei due numeri originali.



PYTHON

```
1 x = int(input('Inserisci x: '))
2 y = int(input('Inserisci y: '))
3 while x != y:
4     if x > y:
5         x = x - y
6     else:
7         y = y - x
8 print(f"Il MCD è {x}")
9 print('Fine :)')
```



ESERCITIAMOCI CON LE VARIABILI

<https://studio.code.org/courses/express-2025/units/1/lessons/27/levels/1>

<https://studio.code.org/courses/express-2025/units/1/lessons/28/levels/1>

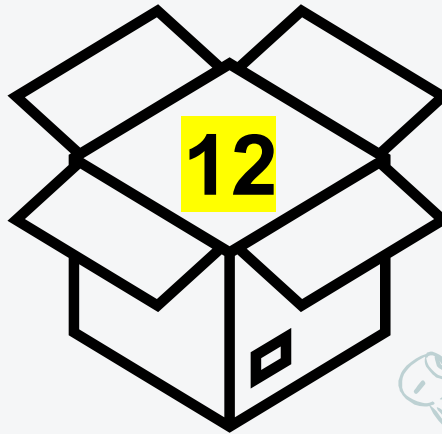
COS'È UNA VARIABILE?

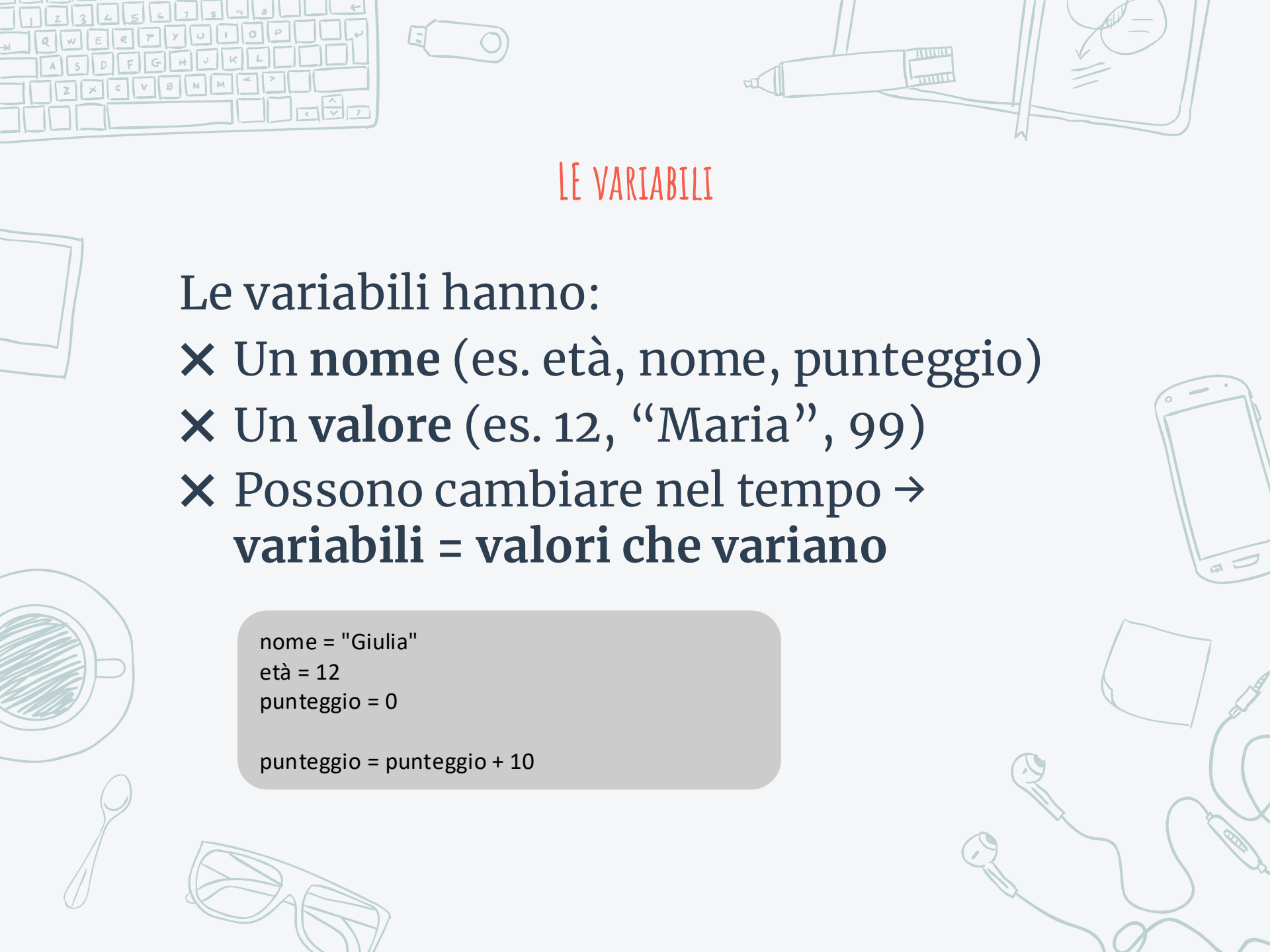
Immagina una scatola con un'etichetta.

- ✗ L'etichetta è il **nome della variabile**.
- ✗ Dentro la scatola metti un **valore**.
- ✗ Puoi aprire la scatola, cambiare il contenuto e richiuderla.

Età

12





LE VARIABILI

Le variabili hanno:

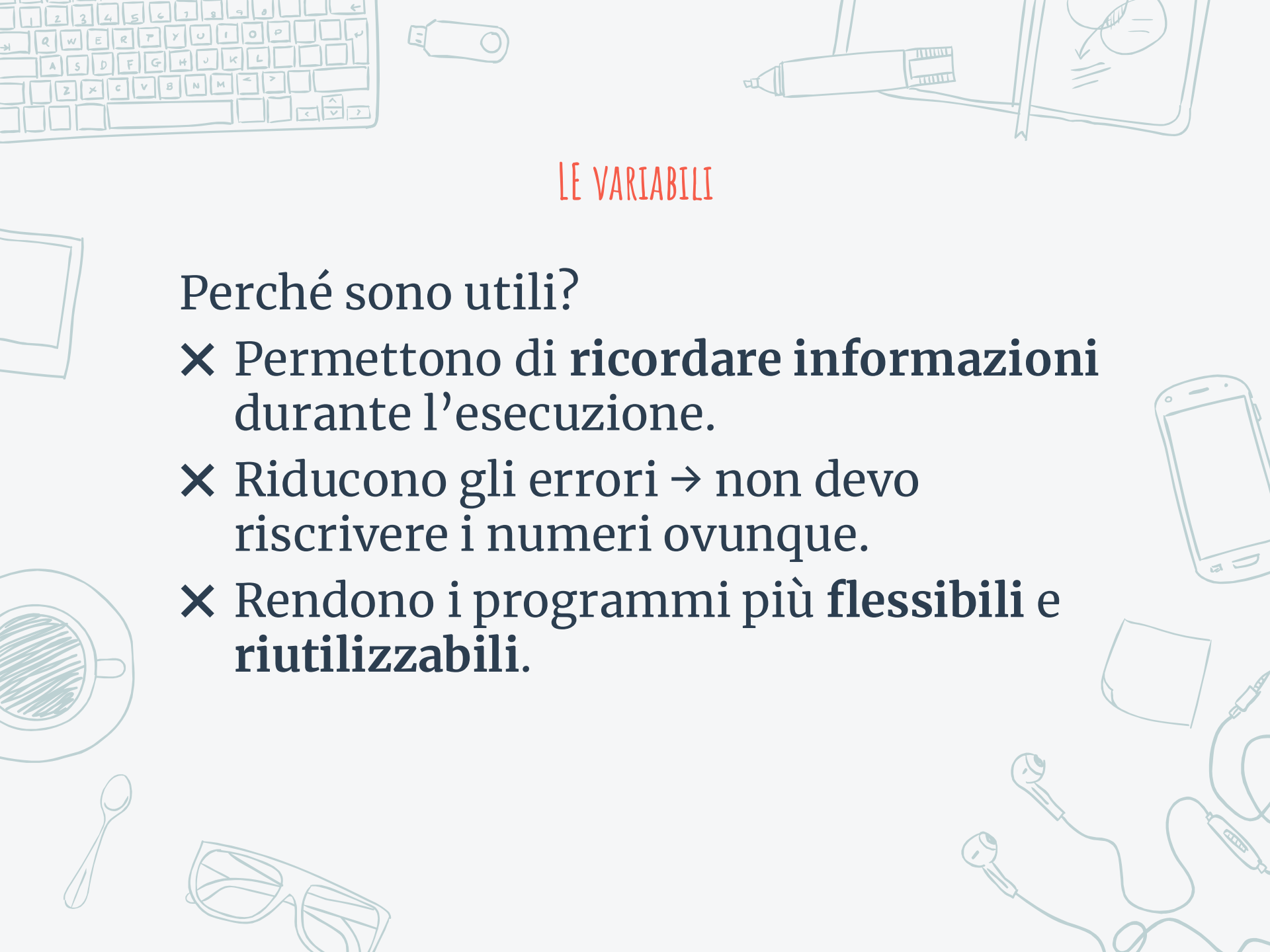
- ✗ Un **nome** (es. età, nome, punteggio)
- ✗ Un **valore** (es. 12, “Maria”, 99)
- ✗ Possono cambiare nel tempo →
variabili = valori che variano

```
nome = "Giulia"
```

```
età = 12
```

```
punteggio = 0
```

```
punteggio = punteggio + 10
```

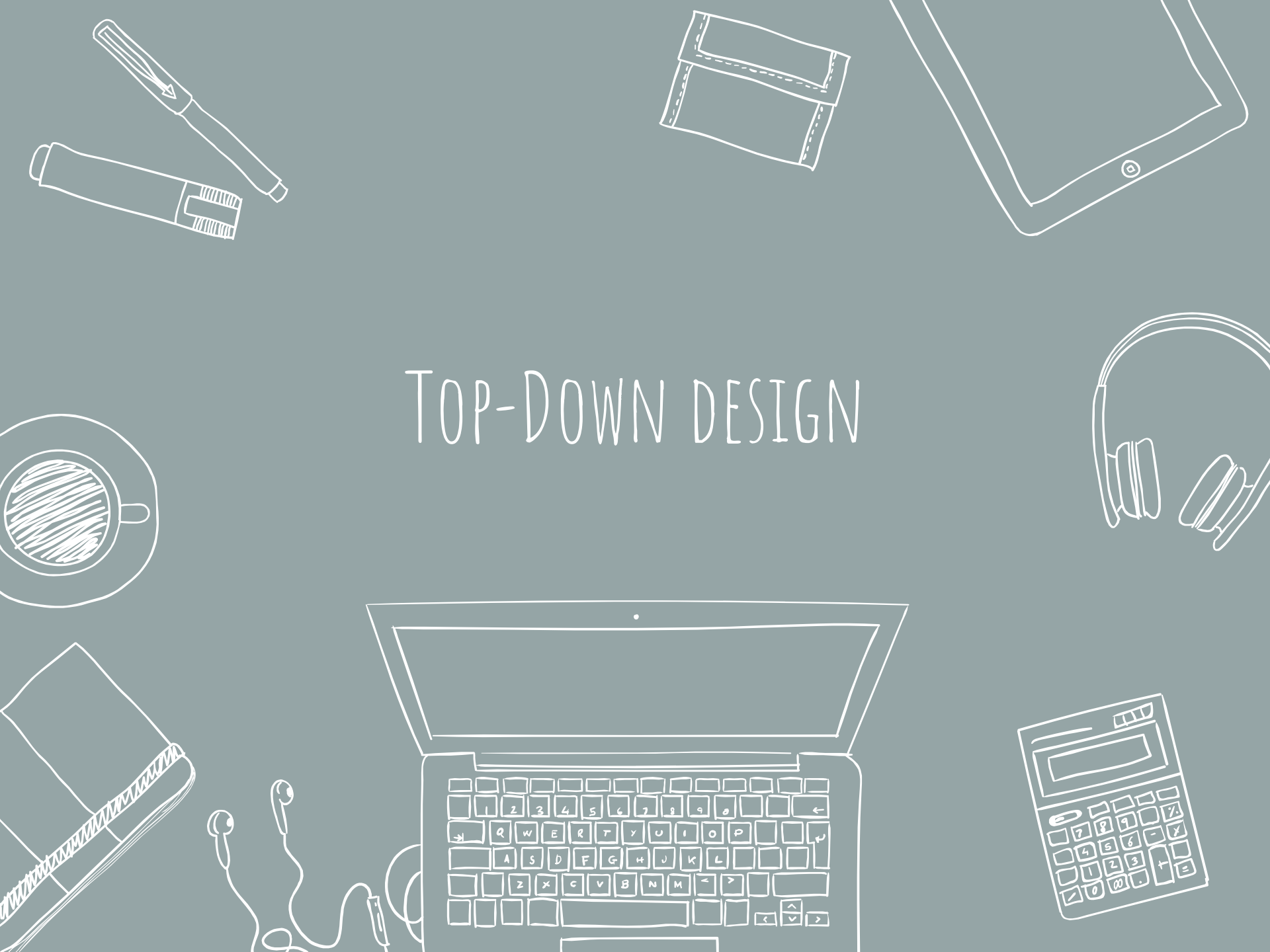


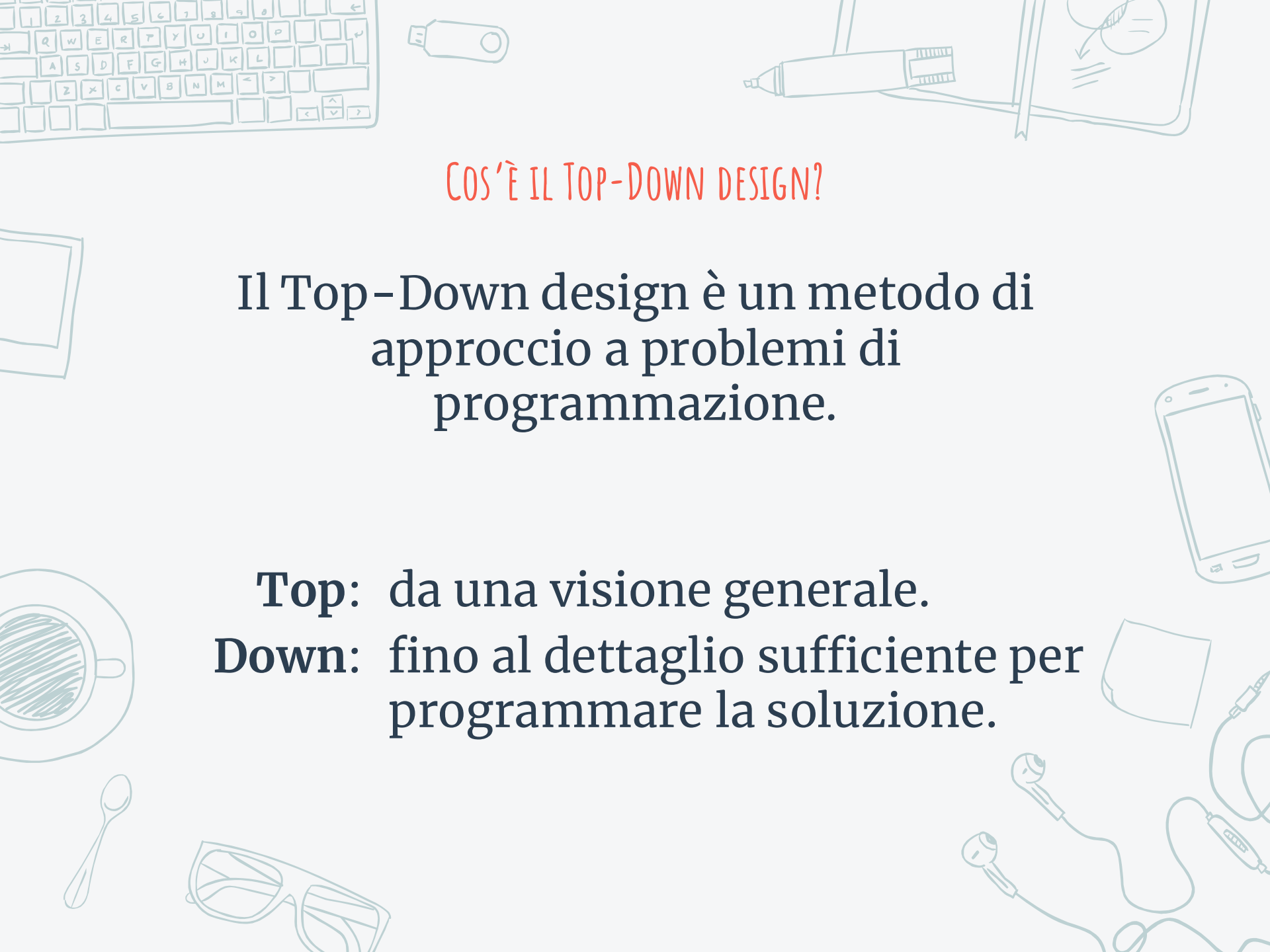
LE VARIABILI

Perché sono utili?

- ✗ Permettono di **ricordare informazioni** durante l'esecuzione.
- ✗ Riducono gli errori → non devo riscrivere i numeri ovunque.
- ✗ Rendono i programmi più **flessibili e riutilizzabili**.

TOP-DOWN DESIGN



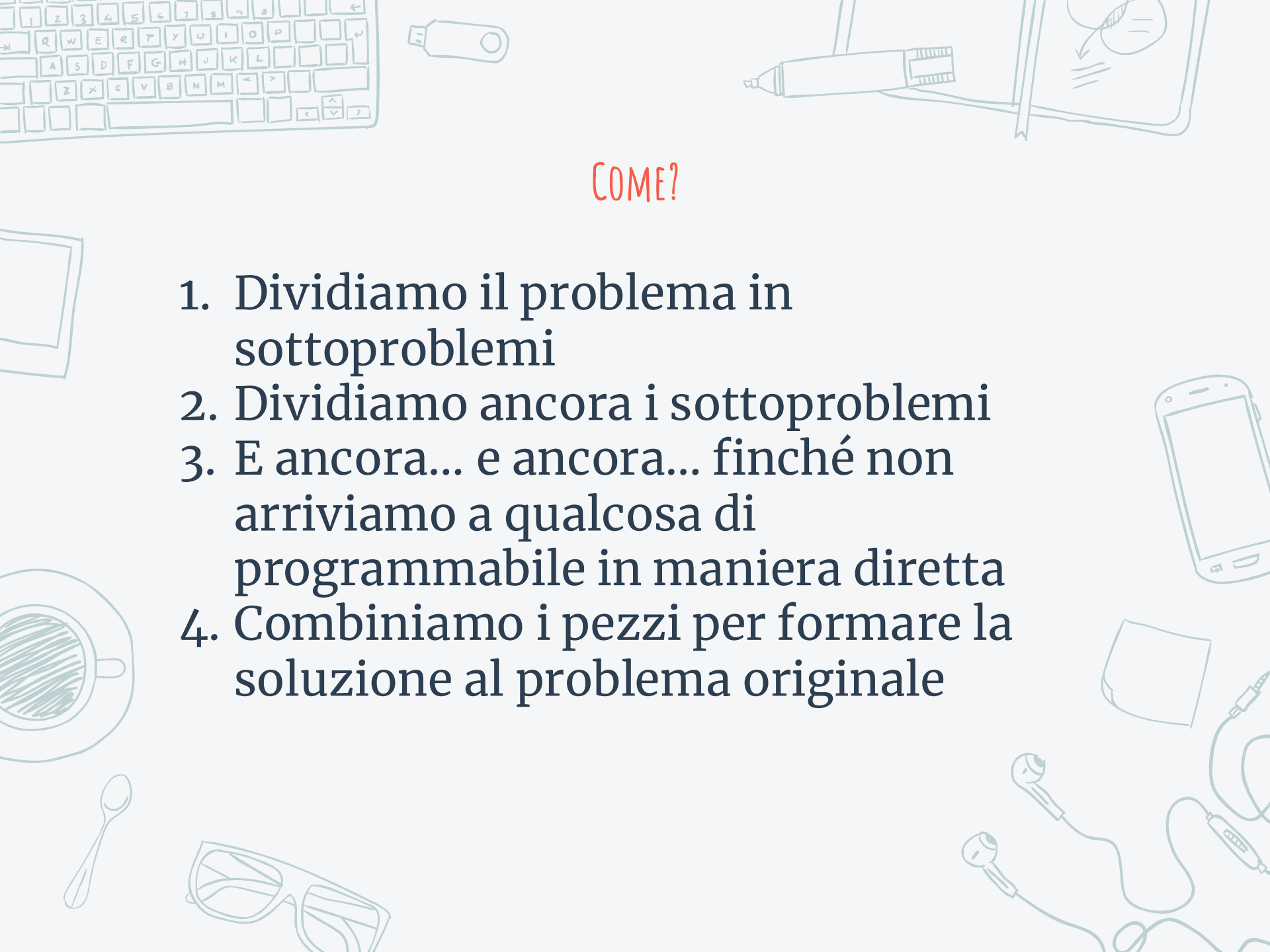


COS'È IL TOP-DOWN DESIGN?

Il Top-Down design è un metodo di approccio a problemi di programmazione.

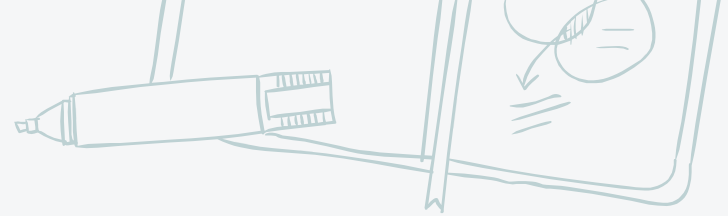
Top: da una visione generale.

Down: fino al dettaglio sufficiente per programmare la soluzione.

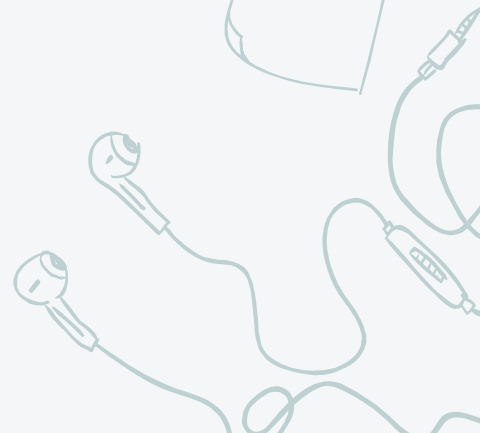
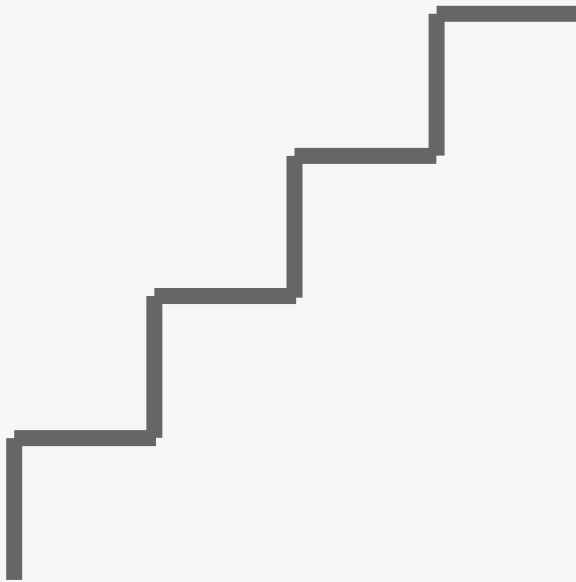


COME?

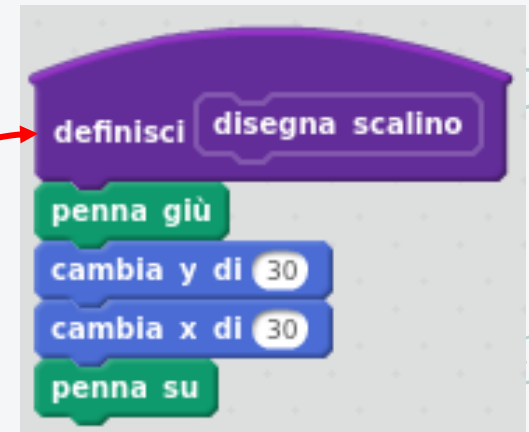
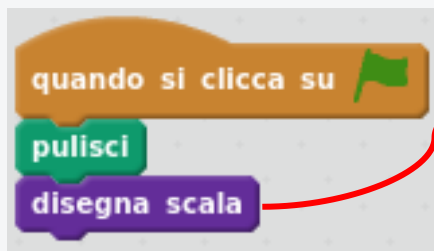
1. Dividiamo il problema in sottoproblemi
2. Dividiamo ancora i sottoproblemi
3. E ancora... e ancora... finché non arriviamo a qualcosa di programmabile in maniera diretta
4. Combiniamo i pezzi per formare la soluzione al problema originale



ESEMPIO

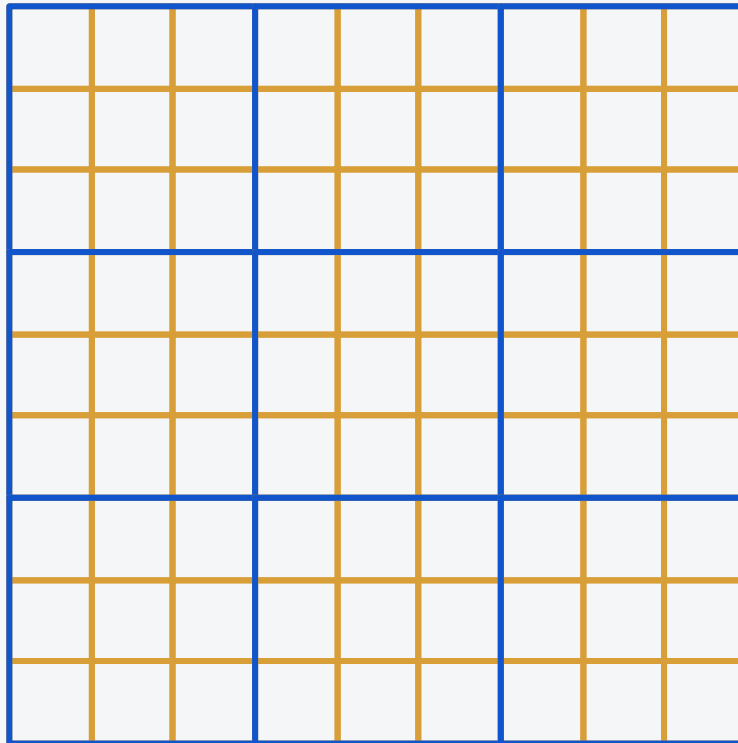


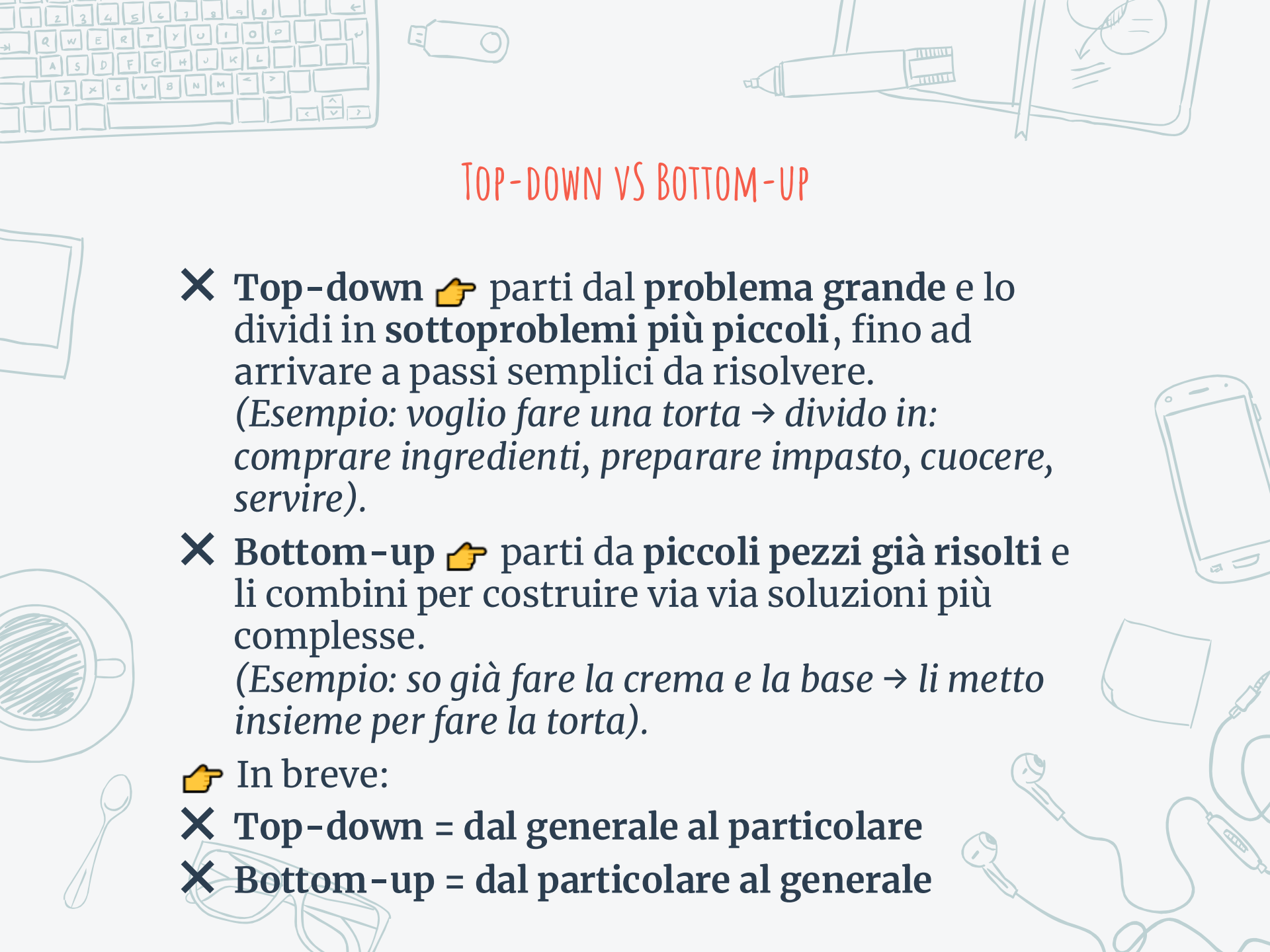
ESEMPIO IN SCRATCH



ESERCIZIO 2

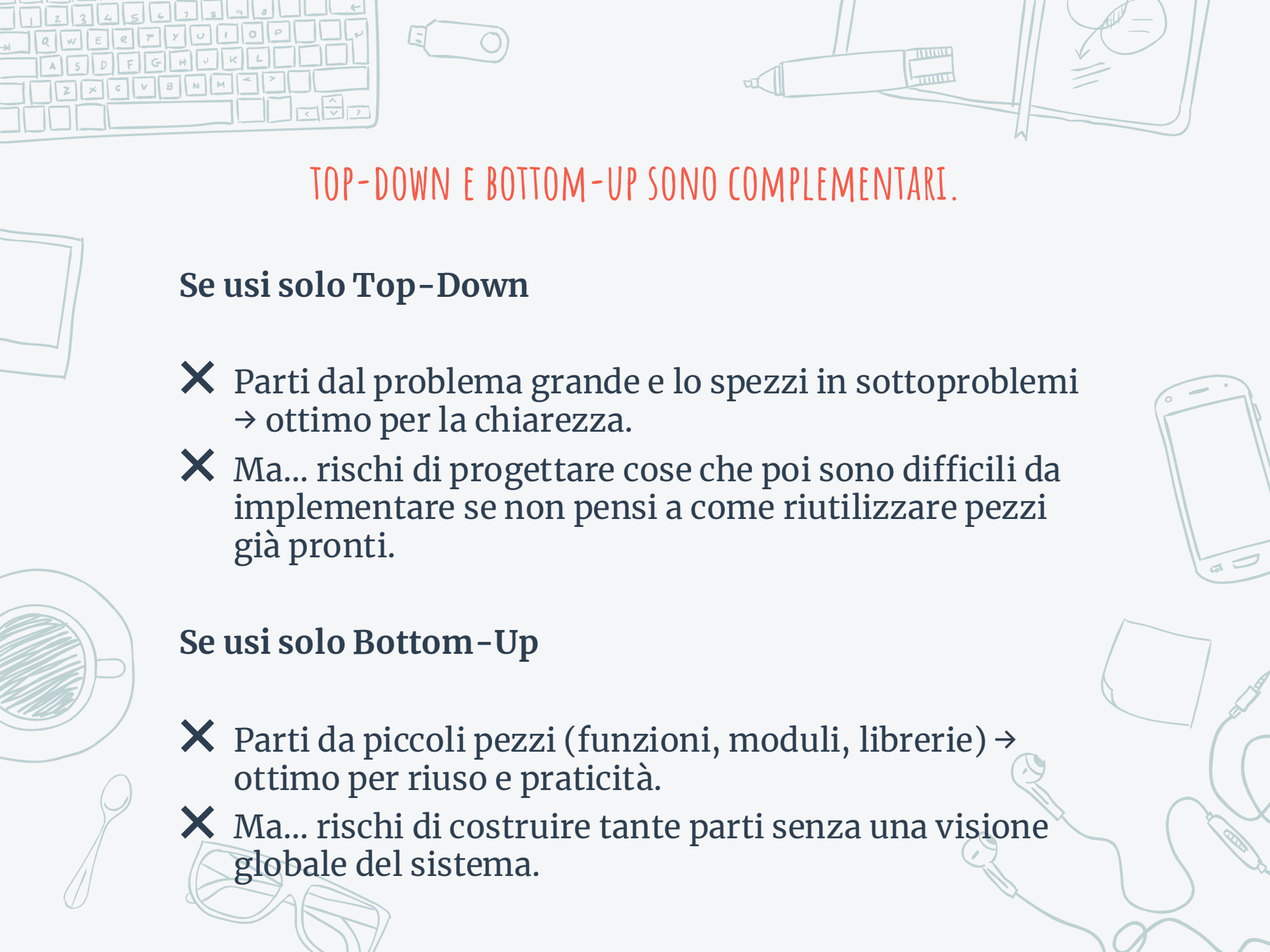
Disegnare una griglia di Sudoku come la seguente (colori compresi):





TOP-DOWN VS BOTTOM-UP

- ✗ **Top-down** 👉 parti dal **problema grande** e lo dividi in **sottoproblemi più piccoli**, fino ad arrivare a passi semplici da risolvere.
(Esempio: voglio fare una torta → divido in: comprare ingredienti, preparare impasto, cuocere, servire).
- ✗ **Bottom-up** 👉 parti da **piccoli pezzi già risolti** e li combini per costruire via via soluzioni più complesse.
(Esempio: so già fare la crema e la base → li metto insieme per fare la torta).
- 👉 In breve:
- ✗ **Top-down** = dal generale al particolare
- ✗ **Bottom-up** = dal particolare al generale



TOP-DOWN E BOTTOM-UP SONO COMPLEMENTARI.

Se usi solo Top-Down

- ✗ Parti dal problema grande e lo spezzi in sottoproblemi → ottimo per la chiarezza.
- ✗ Ma... rischi di progettare cose che poi sono difficili da implementare se non pensi a come riutilizzare pezzi già pronti.

Se usi solo Bottom-Up

- ✗ Parti da piccoli pezzi (funzioni, moduli, librerie) → ottimo per riuso e praticità.
- ✗ Ma... rischi di costruire tante parti senza una visione globale del sistema.



TOP-DOWN E BOTTOM-UP SONO COMPLEMENTARI.

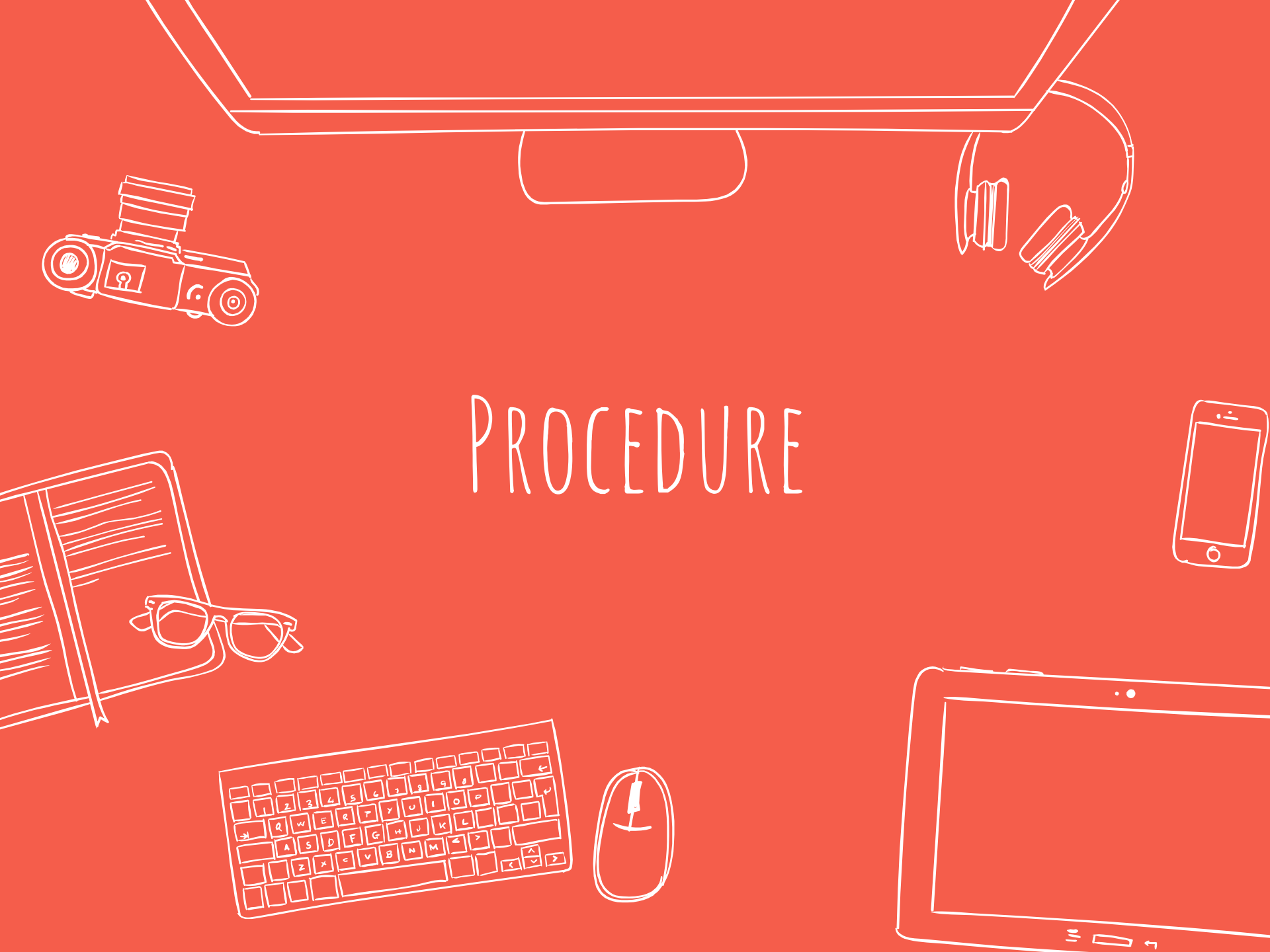
Perché si usano **in momenti diversi**:

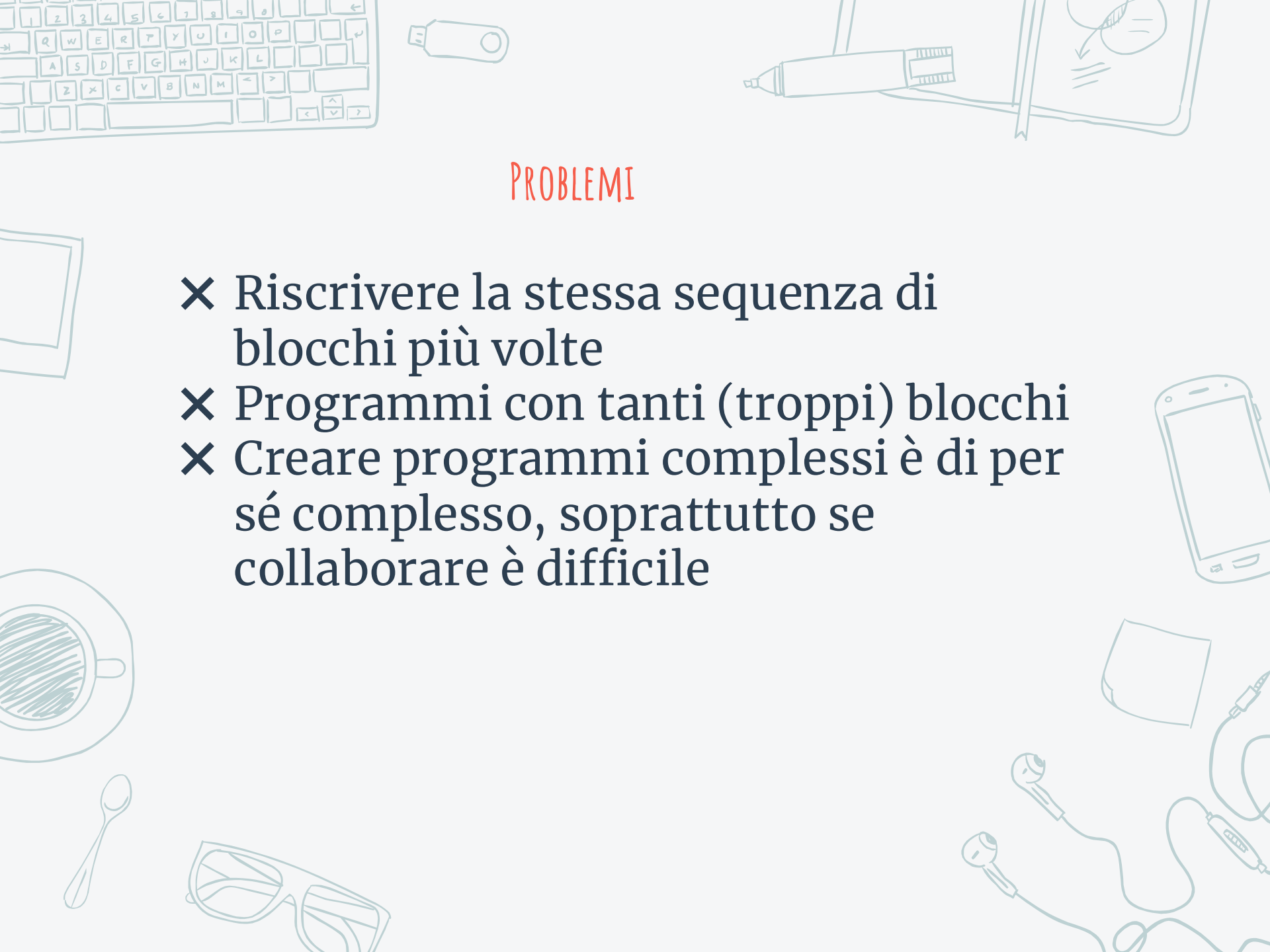
- ✗ **Top-down** serve per capire *cosa* bisogna fare.
- ✗ **Bottom-up** serve per costruire *come* farlo, usando parti già pronte.

In pratica:

- ✗ Progetti un'app **top-down** (disegno architettura, schermate, flusso).
- ✗ La realizzi **bottom-up** (riciclo funzioni per il login, database già fatto, moduli esistenti).

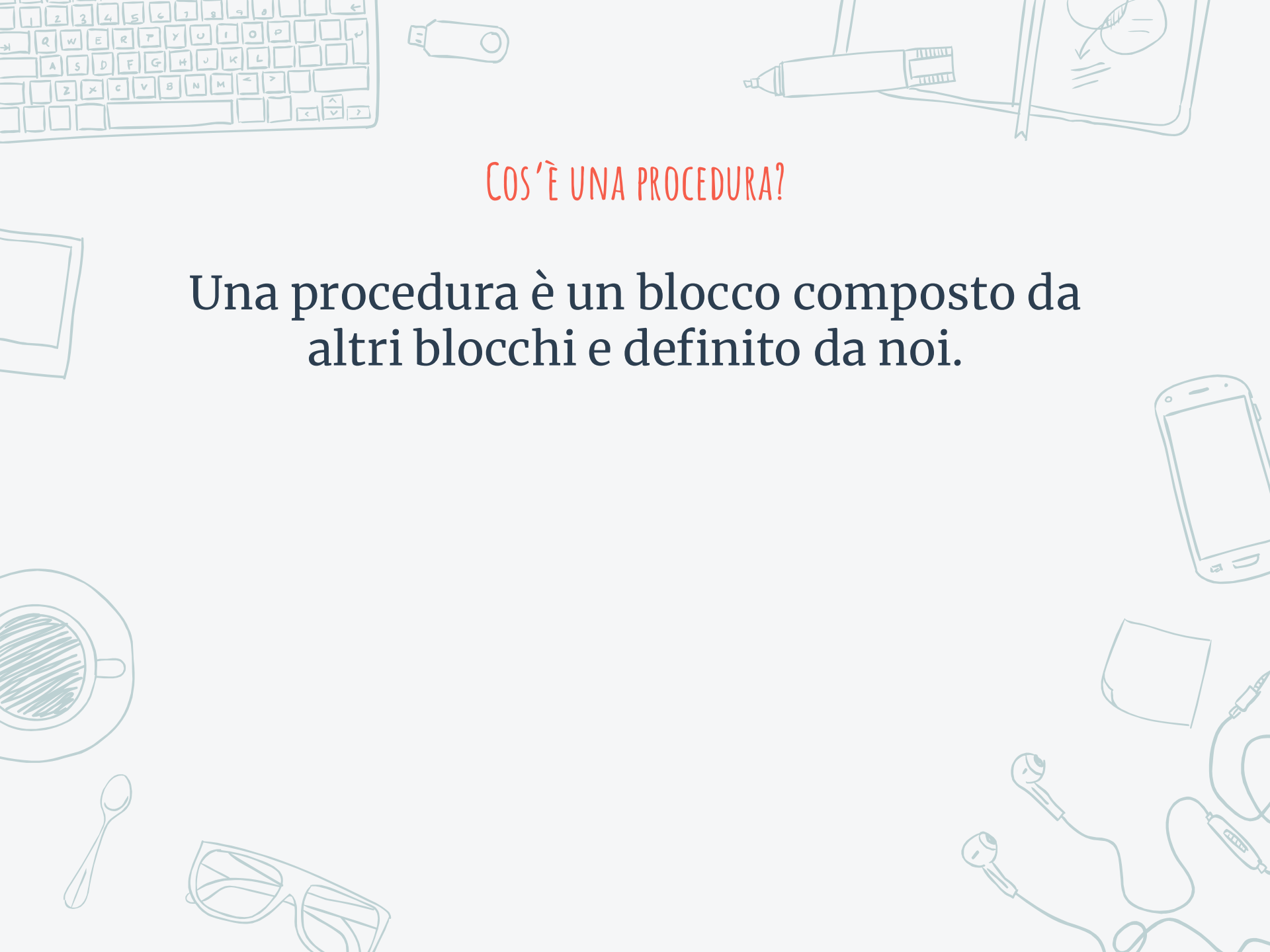
PROCEDURE





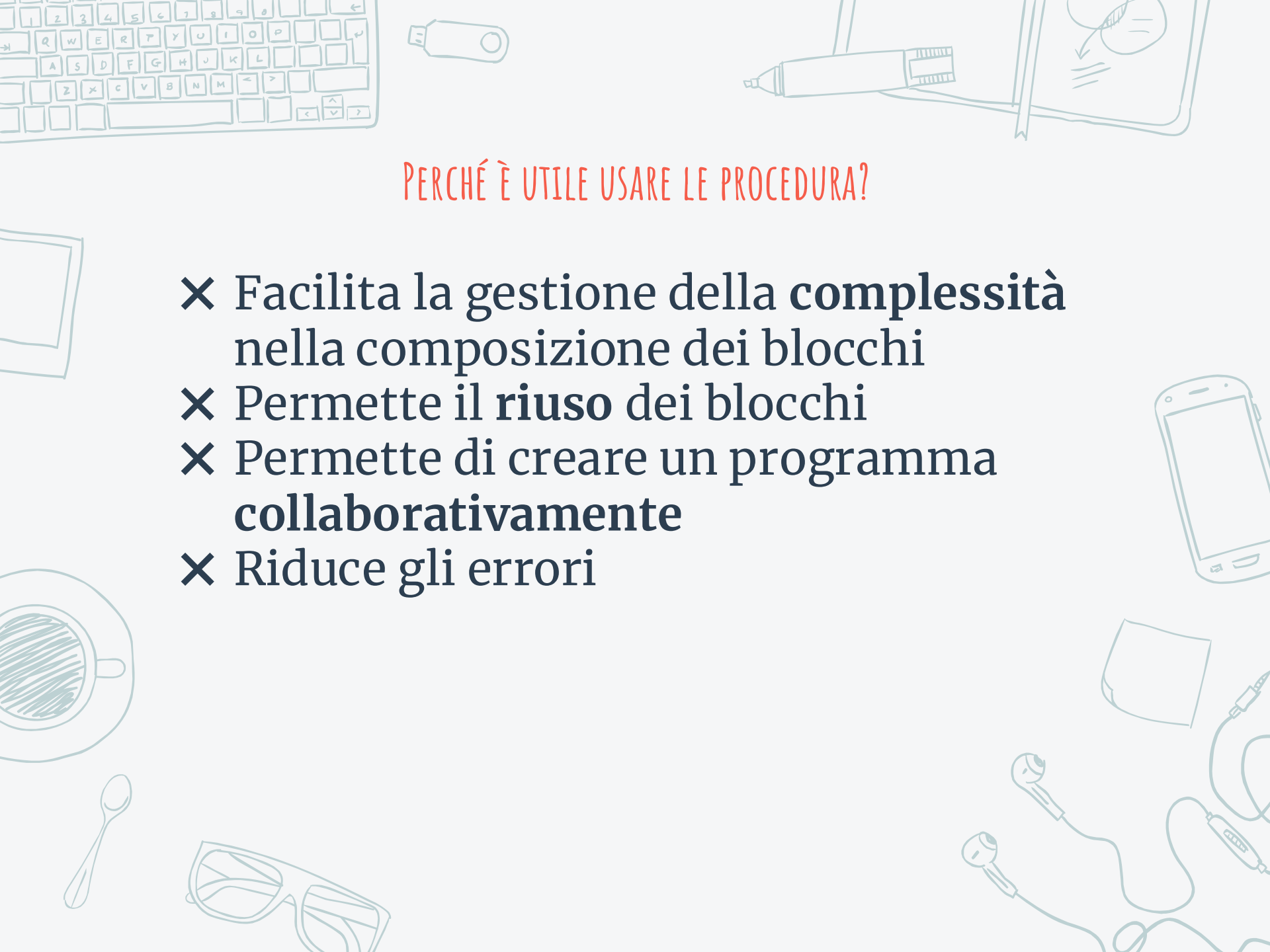
PROBLEMI

- ✗ Riscrivere la stessa sequenza di blocchi più volte
- ✗ Programmi con tanti (troppi) blocchi
- ✗ Creare programmi complessi è di per sé complesso, soprattutto se collaborare è difficile



COS'È UNA PROCEDURA?

Una procedura è un blocco composto da altri blocchi e definito da noi.



PERCHÉ È UTILE USARE LE PROCEDURE?

- ✗ Facilita la gestione della **complessità** nella composizione dei blocchi
- ✗ Permette il **riuso** dei blocchi
- ✗ Permette di creare un programma **collaborativamente**
- ✗ Riduce gli errori

quando si clicca su 

penna giù

cambia x di 50

attendi 1 secondi

cambia y di -50

attendi 1 secondi

cambia x di -50

attendi 1 secondi

cambia y di 50

attendi 1 secondi

penna su

fai 100 passi

attendi 1 secondi

penna giù

cambia x di 50

attendi 1 secondi

cambia y di -50

attendi 1 secondi

cambia x di -50

attendi 1 secondi

cambia y di 50

attendi 1 secondi

penna su

quando si clicca su 

disegna quadrato

fai 100 passi

attendi 1 secondi

disegna quadrato

definisci **disegna quadrato**

penna giù

cambia x di 50

attendi 1 secondi

cambia y di -50

attendi 1 secondi

cambia x di -50

attendi 1 secondi

cambia y di 50

attendi 1 secondi

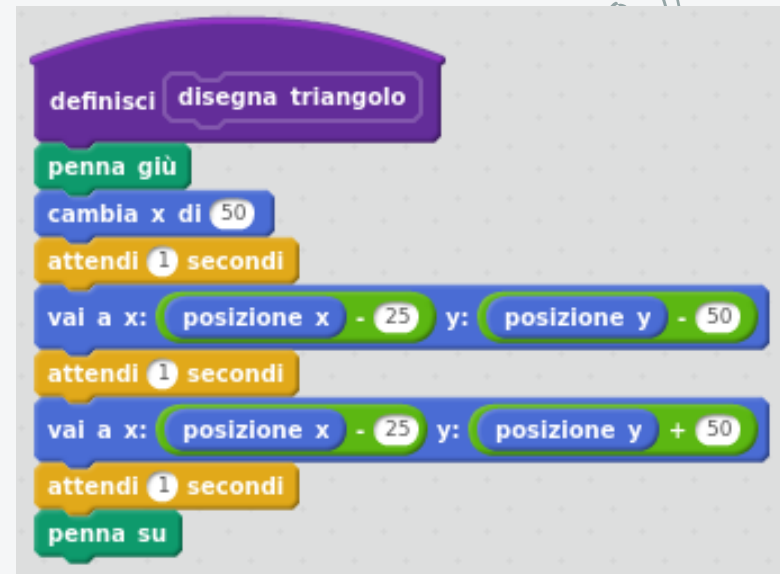
penna su



```
quando si clicca su [bandierina]
disegna quadrato
fai 100 passi
attendi 1 secondi
disegna triangolo
```

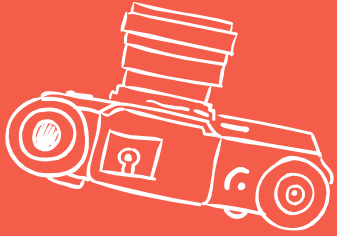


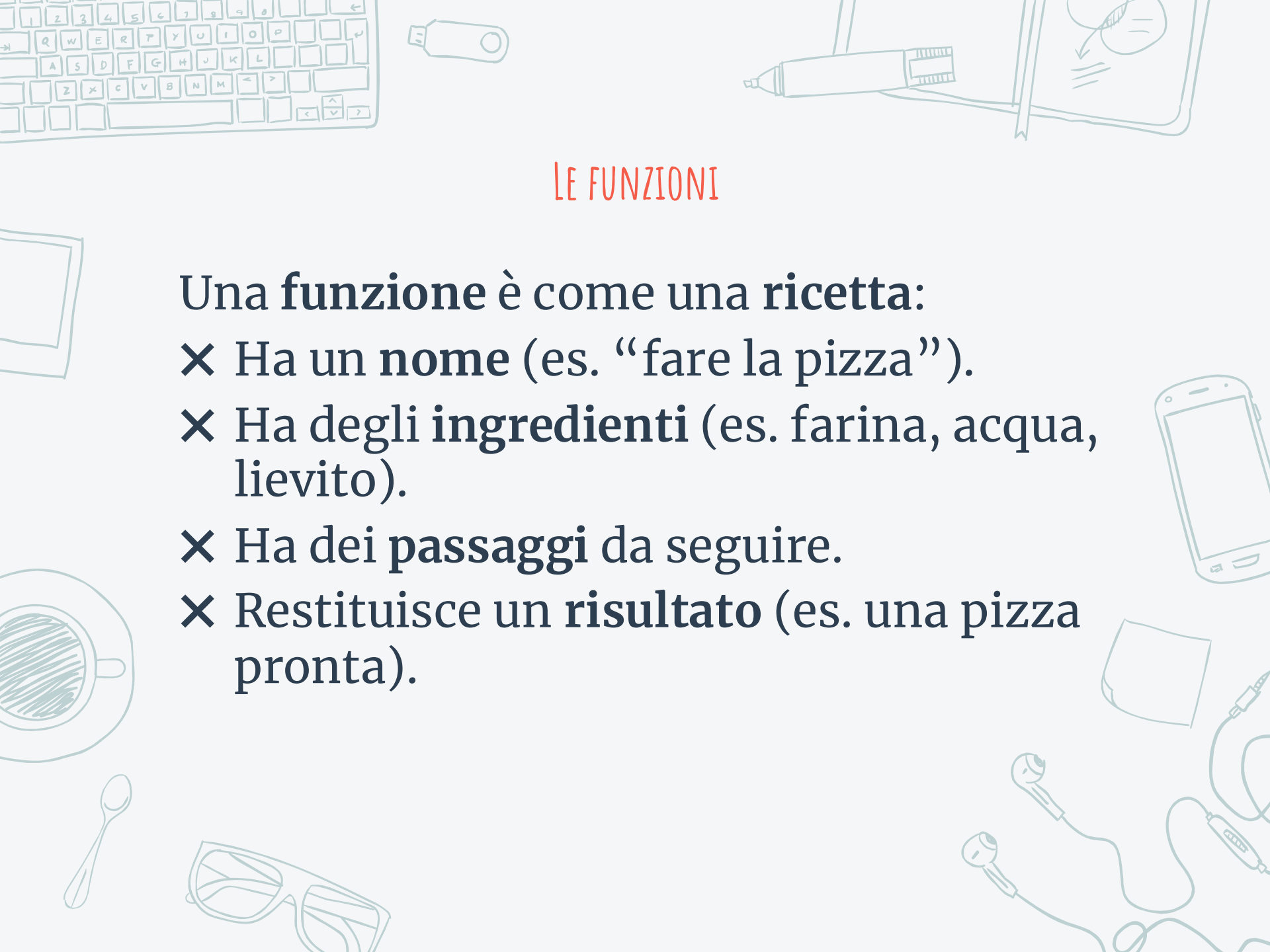
```
definisci [disegna quadrato]
penna giù
cambia x di 50
attendi 1 secondi
cambia y di -50
attendi 1 secondi
cambia x di -50
attendi 1 secondi
cambia y di 50
attendi 1 secondi
penna su
```



```
definisci [disegna triangolo]
penna giù
cambia x di 50
attendi 1 secondi
vai a x: [posizione x - 25] y: [posizione y - 50]
attendi 1 secondi
vai a x: [posizione x - 25] y: [posizione y + 50]
attendi 1 secondi
penna su
```

LE FUNZIONI

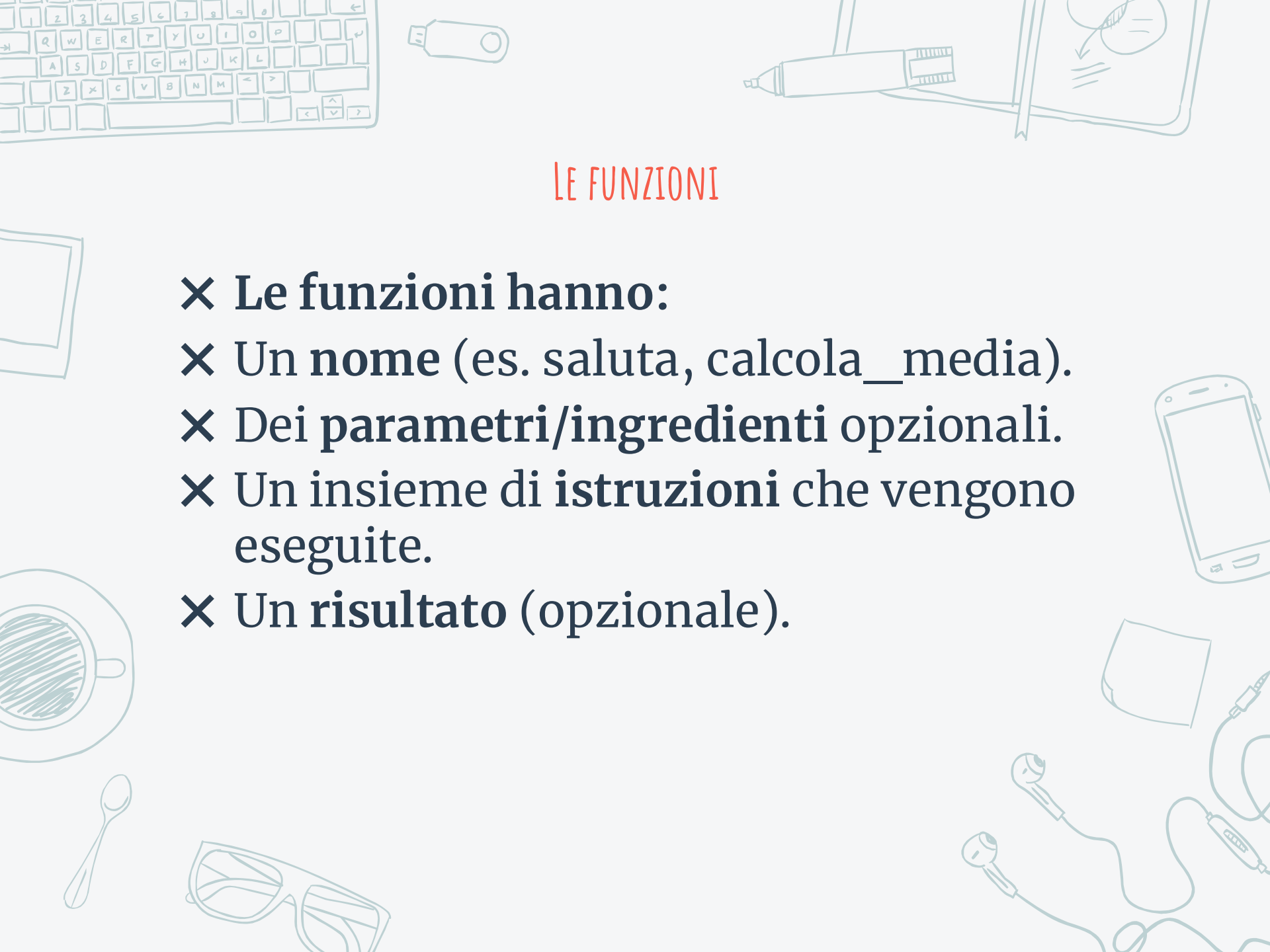




LE FUNZIONI

Una **funzione** è come una **ricetta**:

- ✗ Ha un **nome** (es. “fare la pizza”).
- ✗ Ha degli **ingredienti** (es. farina, acqua, lievito).
- ✗ Ha dei **passaggi** da seguire.
- ✗ Restituisce un **risultato** (es. una pizza pronta).



LE FUNZIONI

- ✗ **Le funzioni hanno:**
- ✗ **Un nome** (es. saluta, calcola_media).
- ✗ **Dei parametri/ingredienti** opzionali.
- ✗ **Un insieme di istruzioni** che vengono eseguite.
- ✗ **Un risultato** (opzionale).

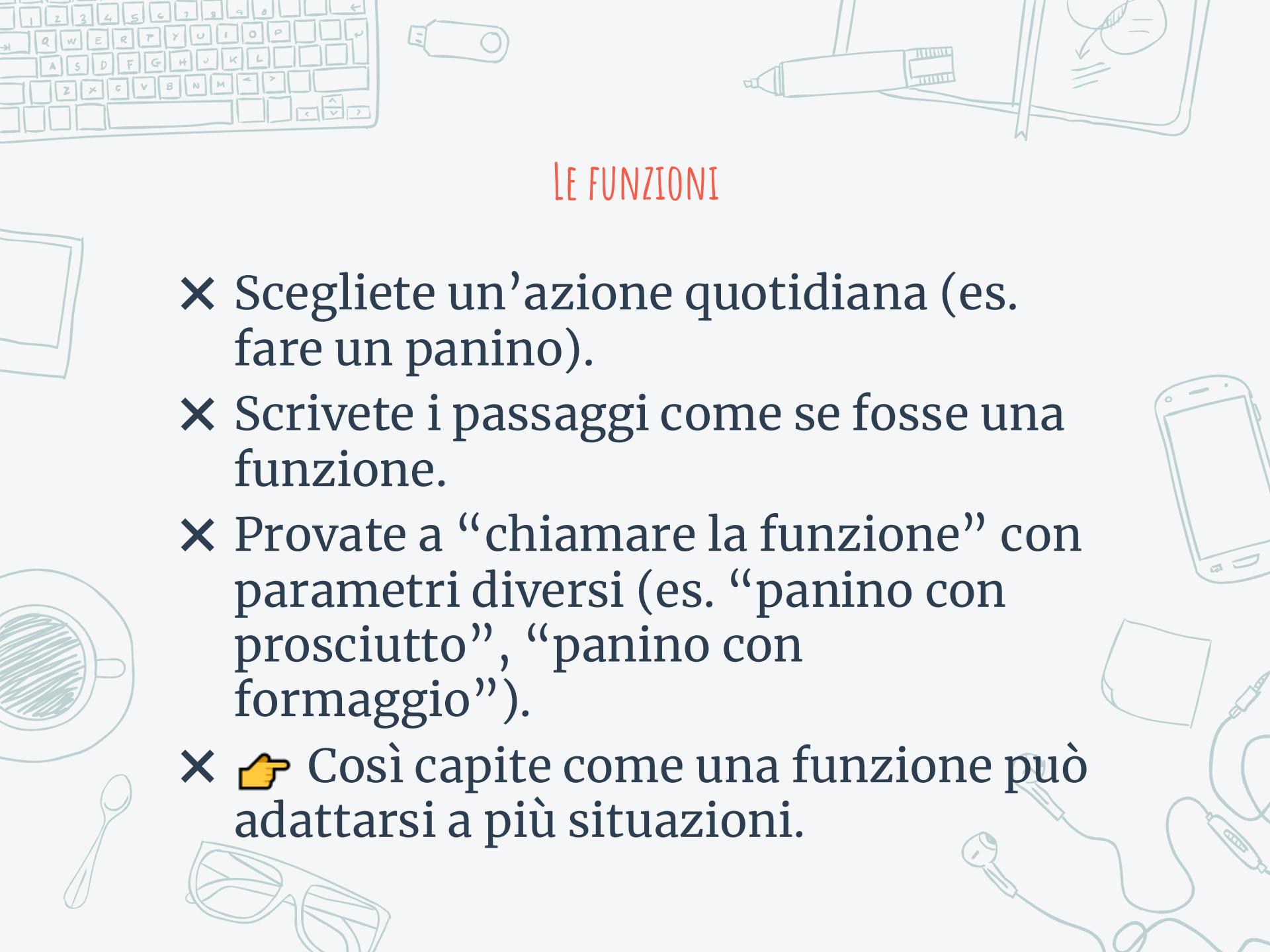


LE FUNZIONI

Funzioni nella vita reale:

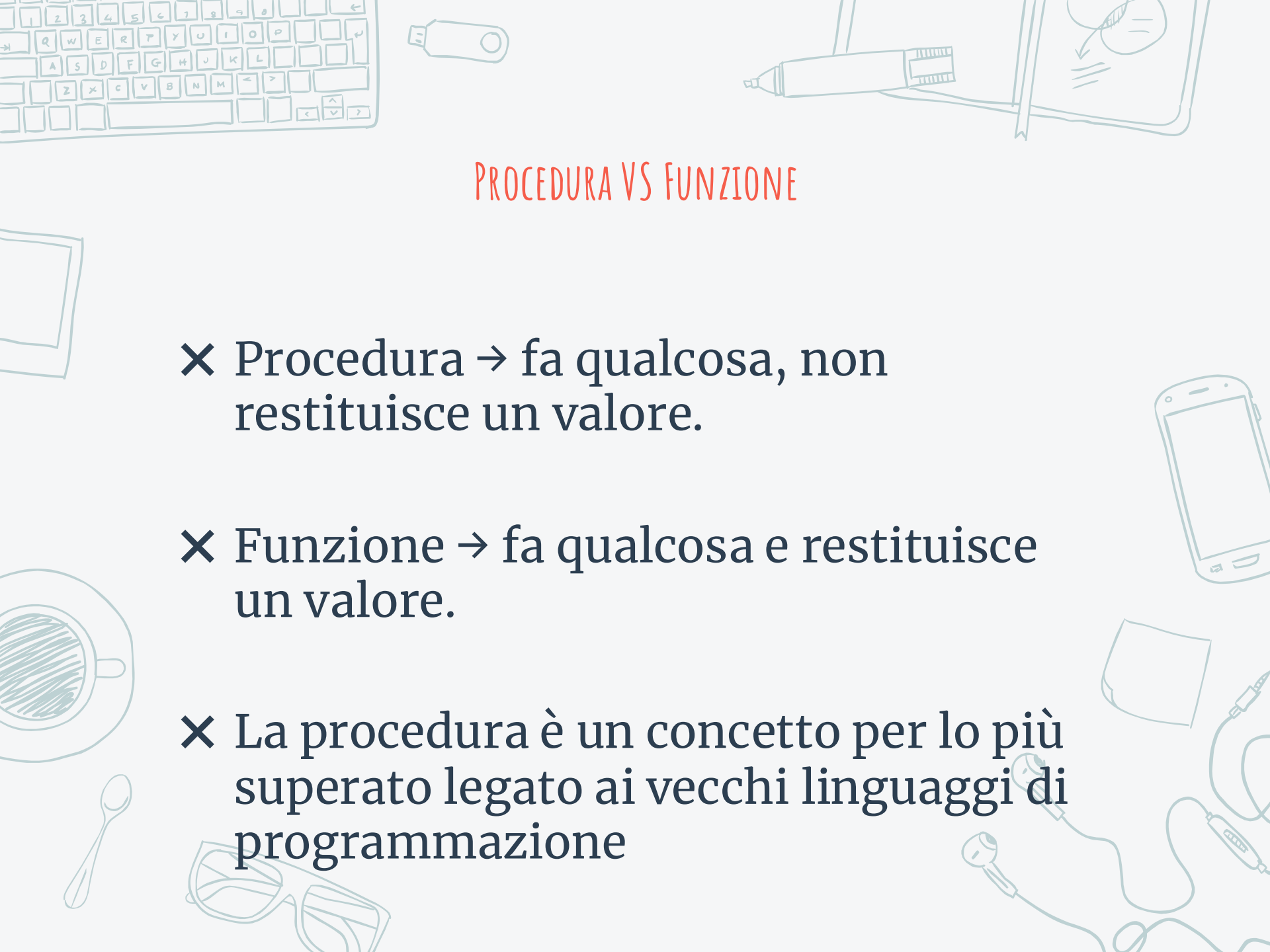
- ✗ “Allacciare le scarpe” → prendi scarpa, lacci, fai i nodi → risultato: scarpa allacciata.
- ✗ “Accendere la luce” → premi l’interruttore → risultato: stanza illuminata.
- ✗ “Fare una somma” → metti due numeri → ottieni il totale.

```
funzione saluta(nome):  
    stampa("Ciao " + nome)  
  
saluta("Marco")  
saluta("Giulia")
```



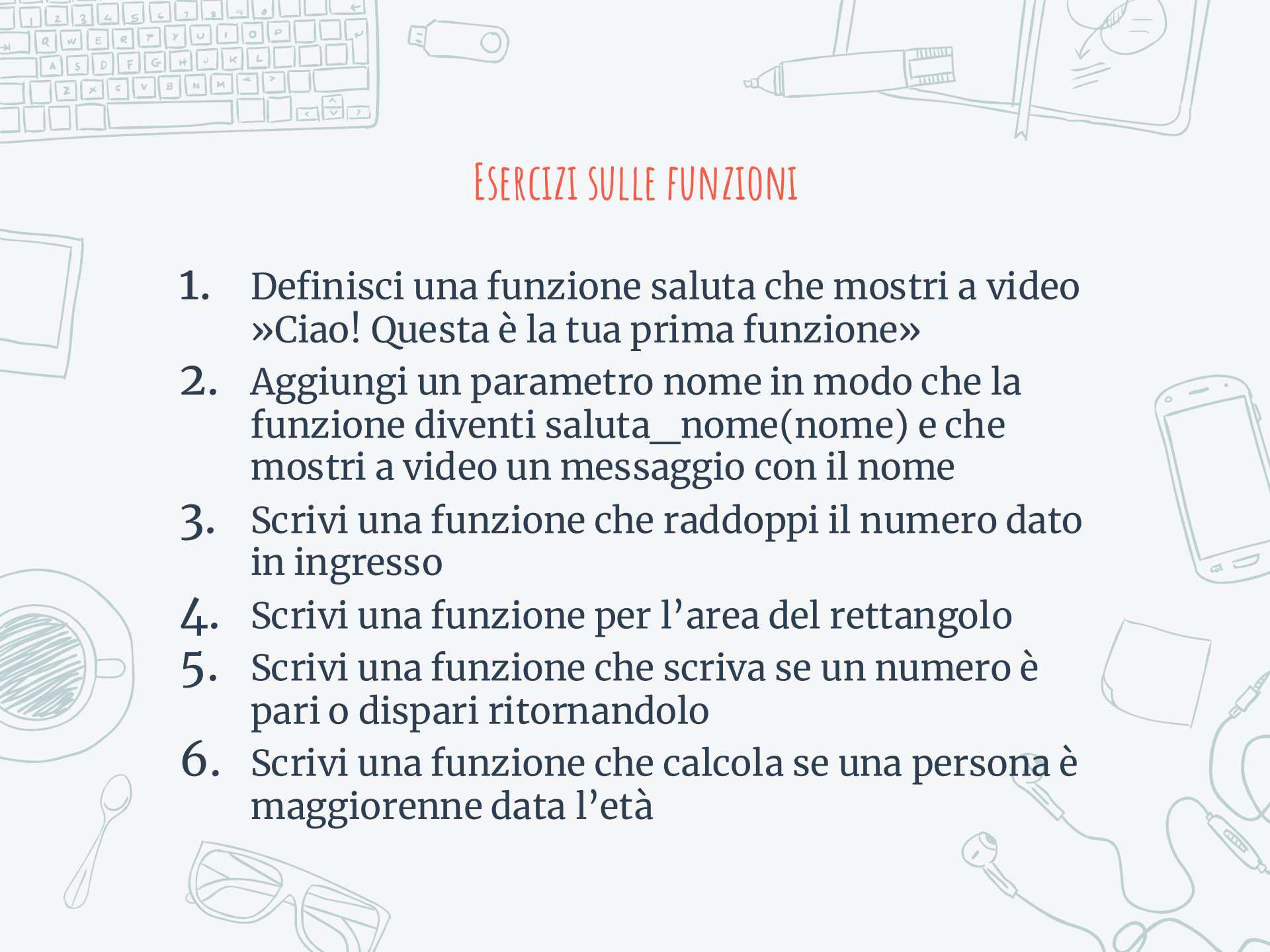
LE FUNZIONI

- ✗ Scegliete un'azione quotidiana (es. fare un panino).
- ✗ Scrivete i passaggi come se fosse una funzione.
- ✗ Provate a “chiamare la funzione” con parametri diversi (es. “panino con prosciutto”, “panino con formaggio”).
- ✗  Così capite come una funzione può adattarsi a più situazioni.



PROCEDURA VS FUNZIONE

- ✗ Procedura → fa qualcosa, non restituisce un valore.
- ✗ Funzione → fa qualcosa e restituisce un valore.
- ✗ La procedura è un concetto per lo più superato legato ai vecchi linguaggi di programmazione



ESERCIZI SULLE FUNZIONI

1. Definisci una funzione saluta che mostri a video
»Ciao! Questa è la tua prima funzione«
2. Aggiungi un parametro nome in modo che la
funzione diventi saluta_nome(nome) e che
mostri a video un messaggio con il nome
3. Scrivi una funzione che raddoppi il numero dato
in ingresso
4. Scrivi una funzione per l'area del rettangolo
5. Scrivi una funzione che scriva se un numero è
pari o dispari ritornandolo
6. Scrivi una funzione che calcola se una persona è
maggiorenne data l'età

ESERCIZIO 5

Dati due numeri interi positivi x e y calcolare il minimo comune multiplo tra x e y usando somma e sottrazione.

