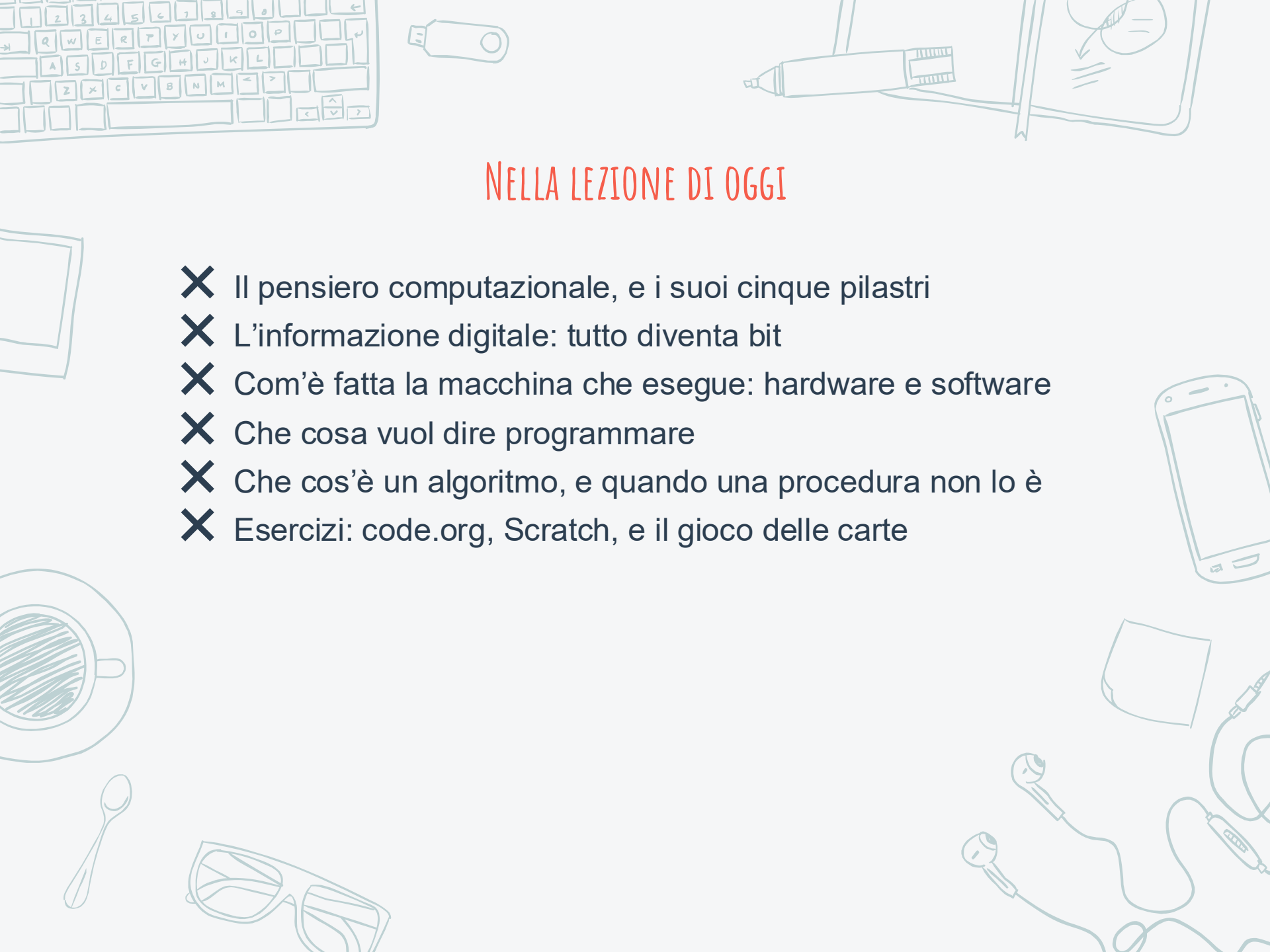




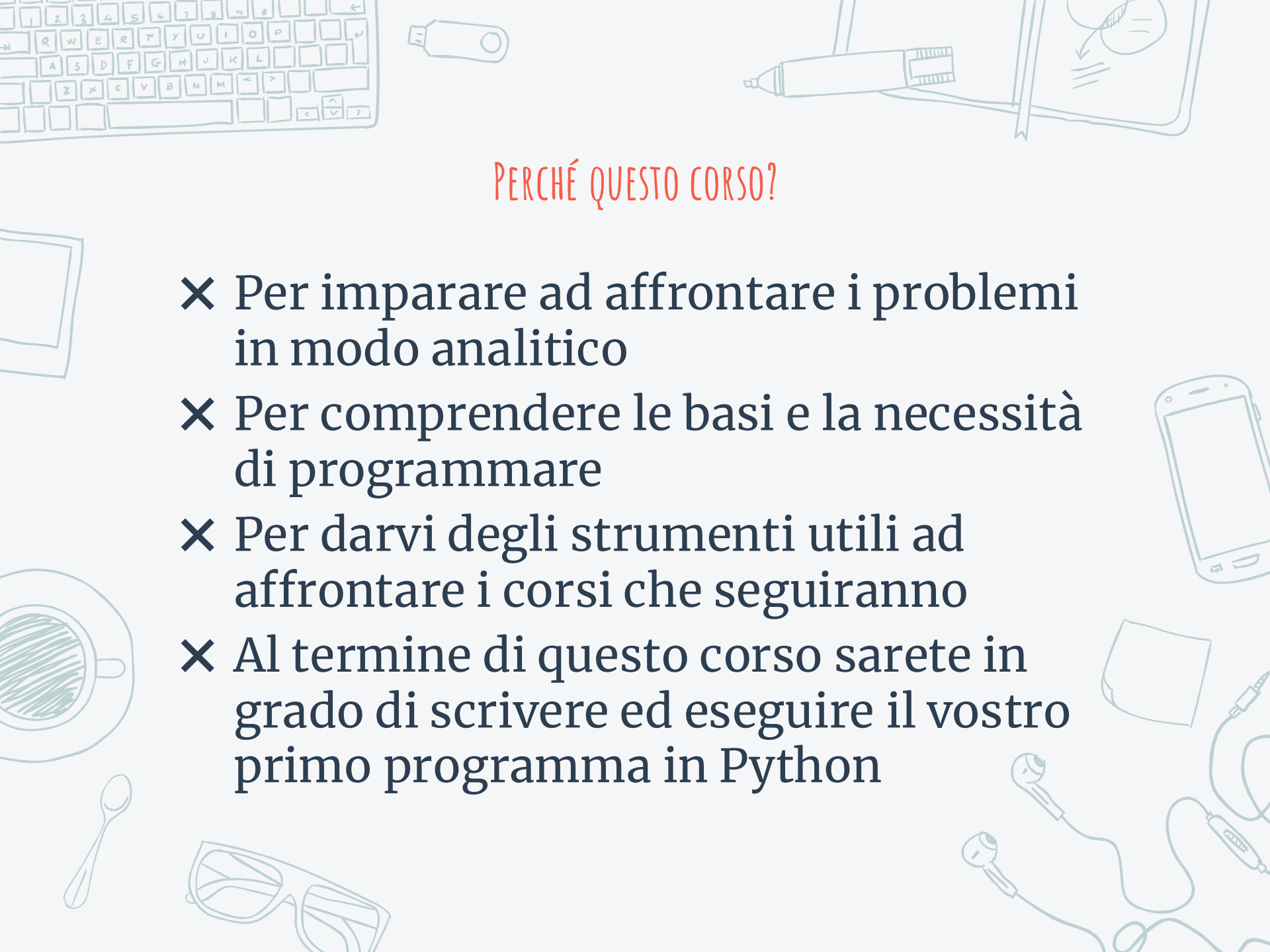
LABORATORIO DI PENSIERO COMPUTAZIONALE

Simone Zennaro – simone.zennaro@unipd.it



NELLA LEZIONE DI OGGI

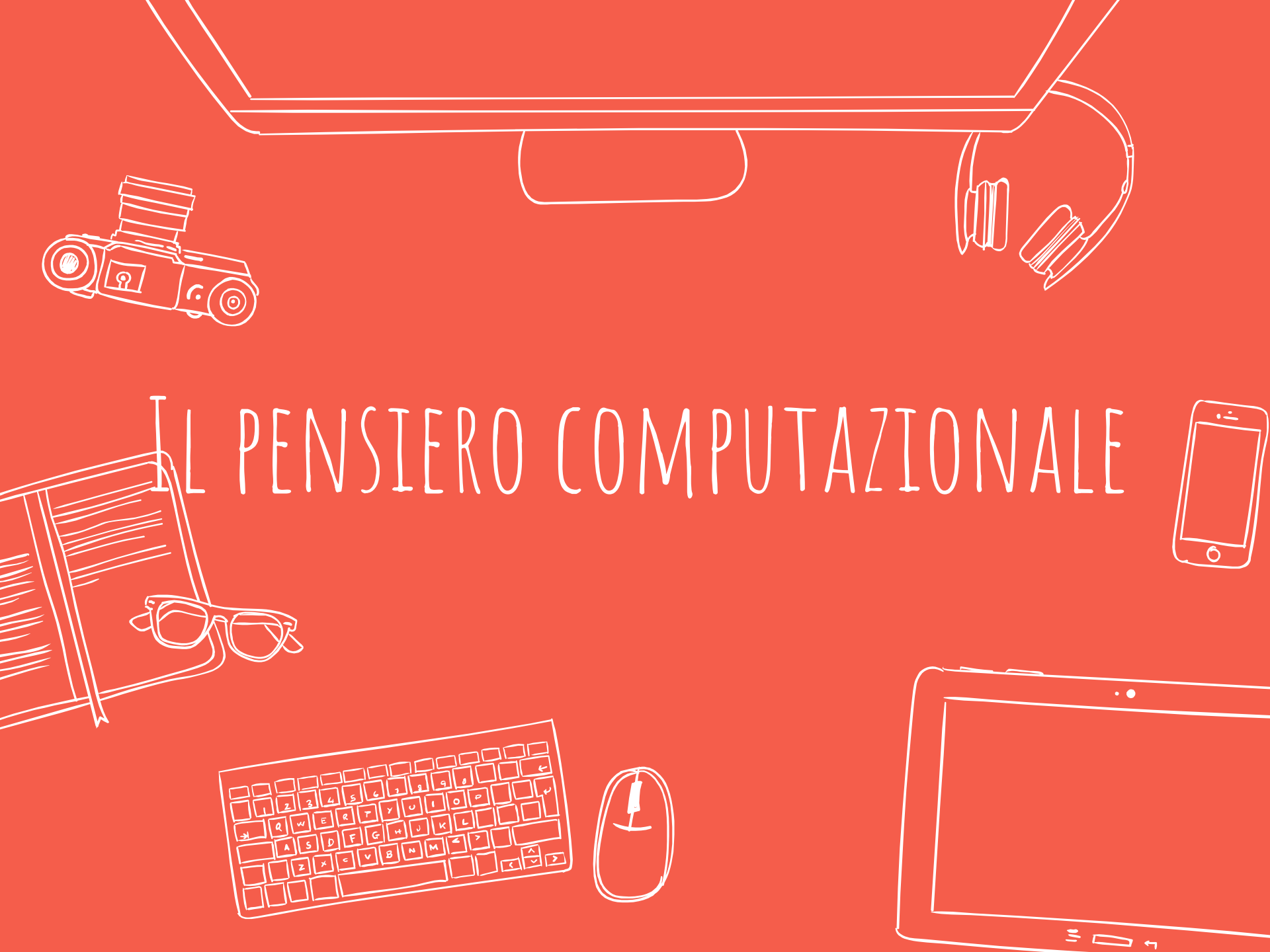
- ✗ Il pensiero computazionale, e i suoi cinque pilastri
- ✗ L'informazione digitale: tutto diventa bit
- ✗ Com'è fatta la macchina che esegue: hardware e software
- ✗ Che cosa vuol dire programmare
- ✗ Che cos'è un algoritmo, e quando una procedura non lo è
- ✗ Esercizi: code.org, Scratch, e il gioco delle carte

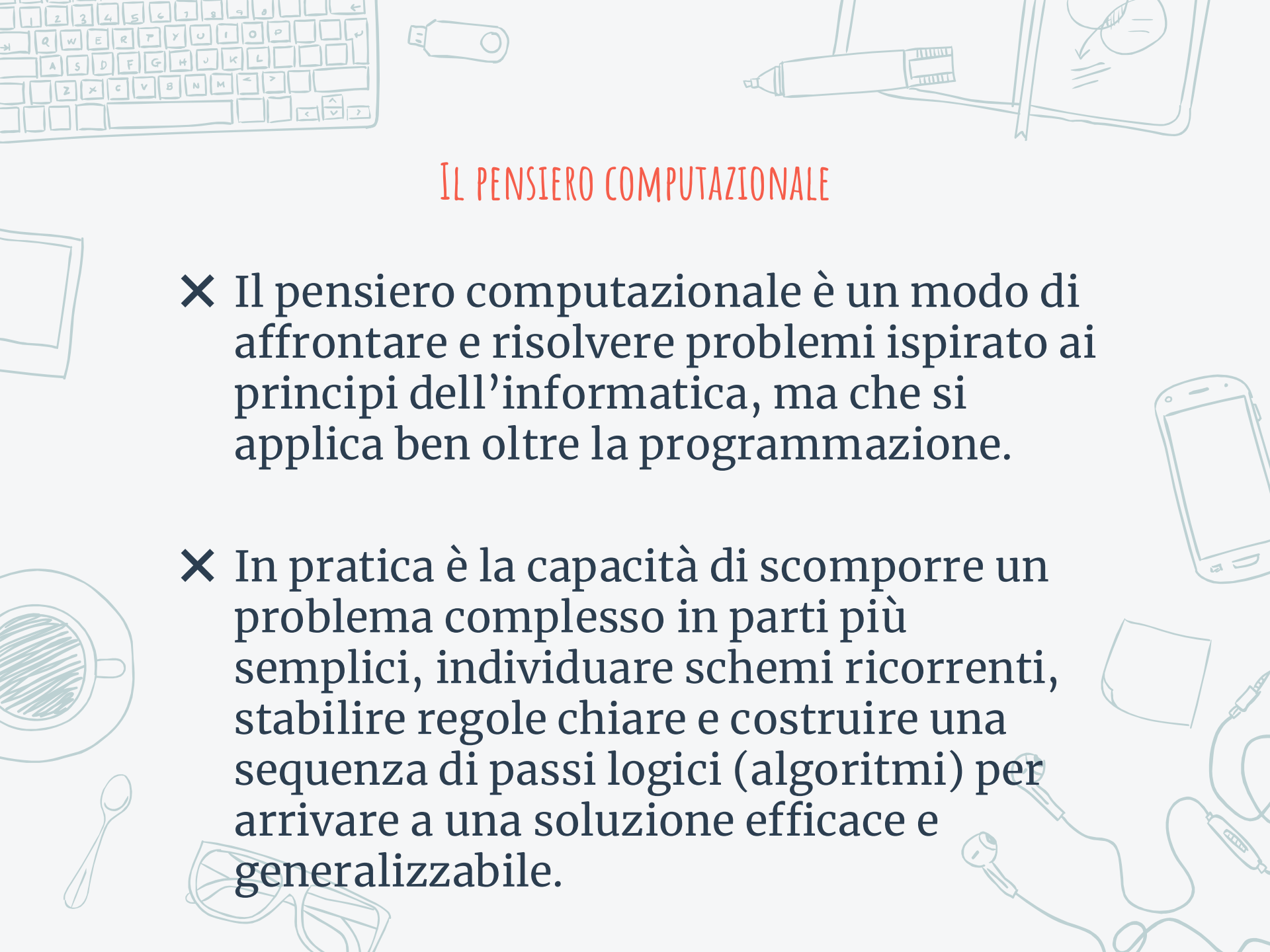


PERCHÉ QUESTO CORSO?

- ✗ Per imparare ad affrontare i problemi in modo analitico
- ✗ Per comprendere le basi e la necessità di programmare
- ✗ Per darvi degli strumenti utili ad affrontare i corsi che seguiranno
- ✗ Al termine di questo corso sarete in grado di scrivere ed eseguire il vostro primo programma in Python

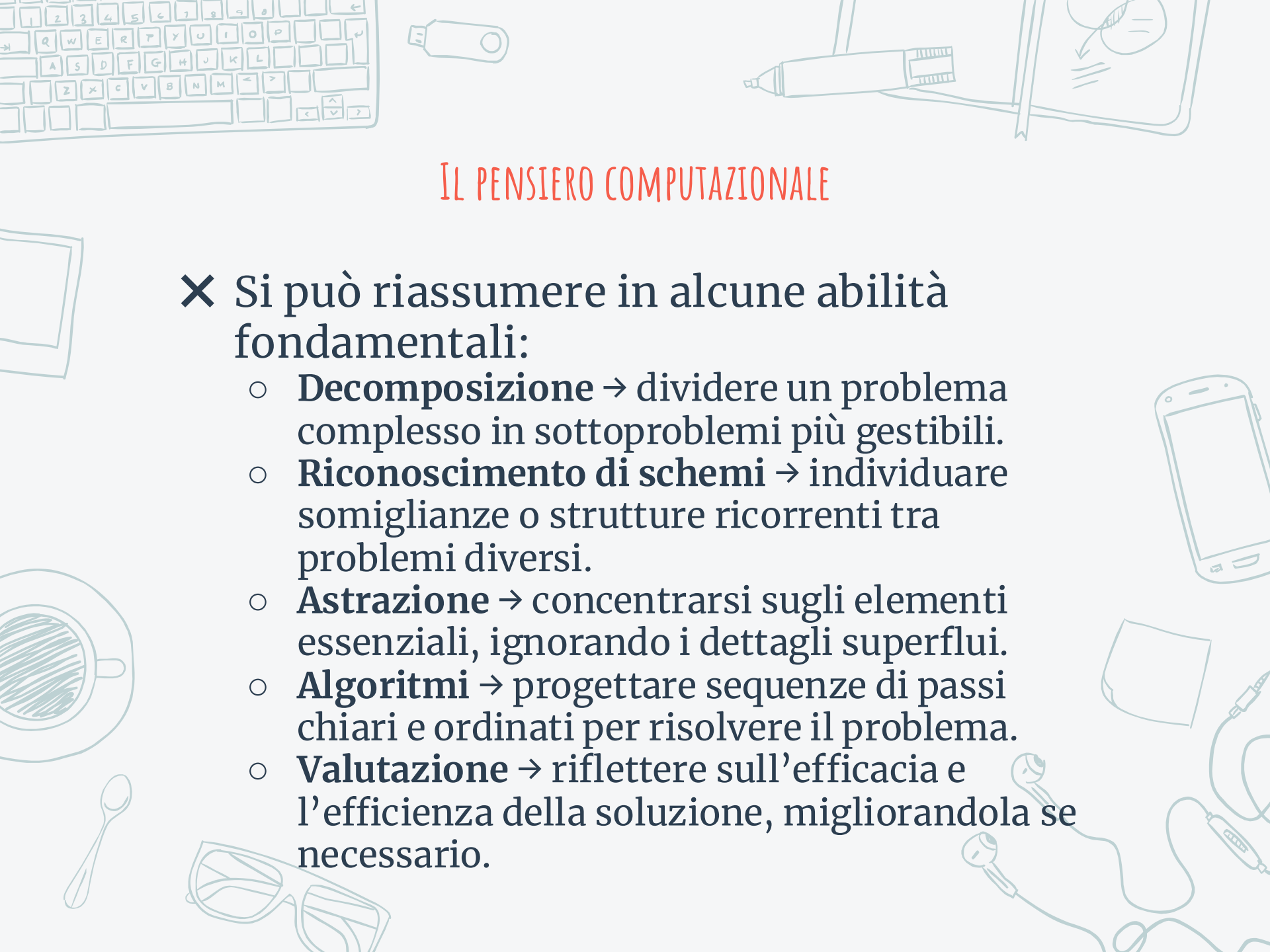
IL PENSIERO COMPUTAZIONALE





IL PENSIERO COMPUTAZIONALE

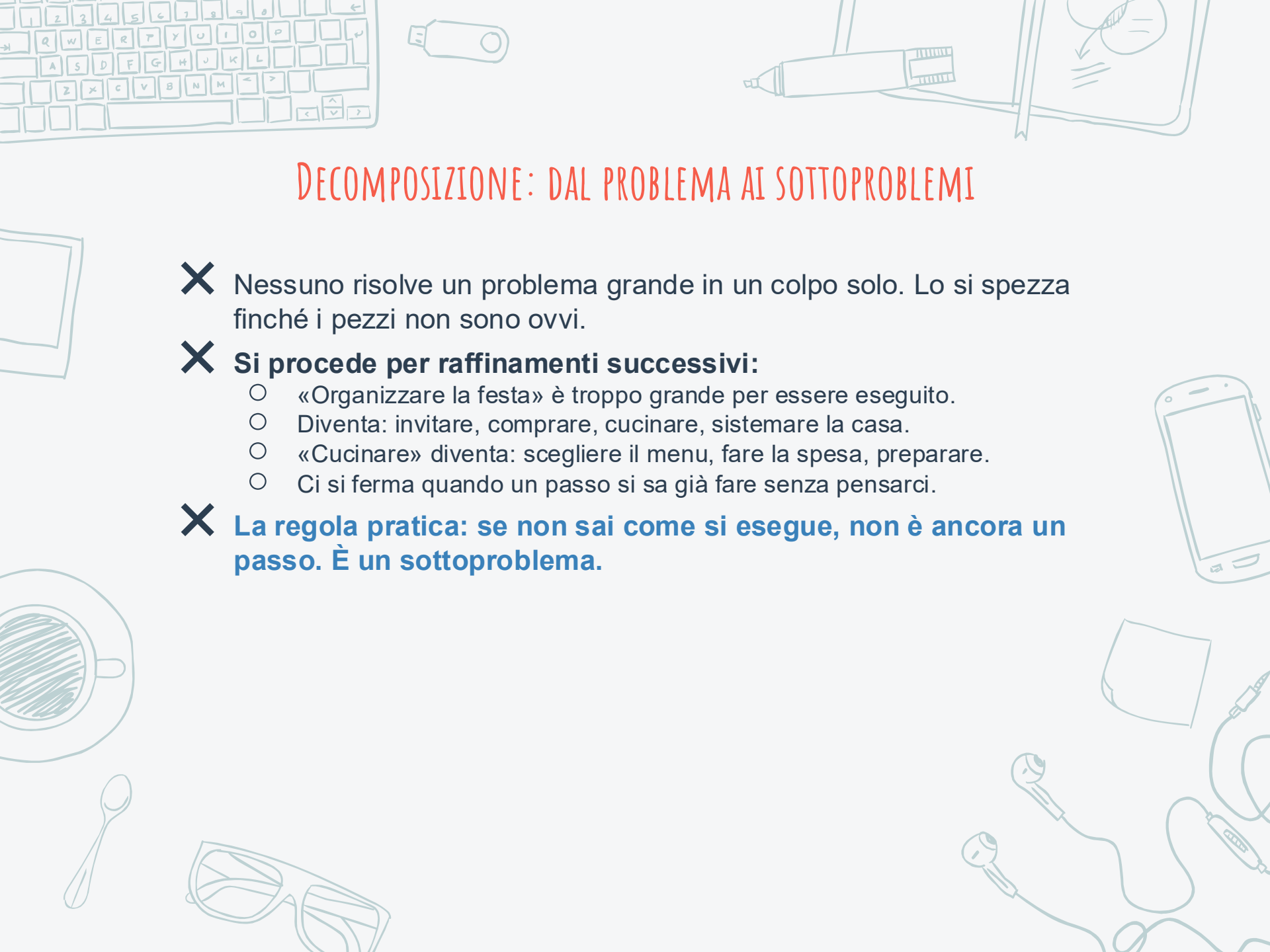
- ✗ Il pensiero computazionale è un modo di affrontare e risolvere problemi ispirato ai principi dell'informatica, ma che si applica ben oltre la programmazione.
- ✗ In pratica è la capacità di scomporre un problema complesso in parti più semplici, individuare schemi ricorrenti, stabilire regole chiare e costruire una sequenza di passi logici (algoritmi) per arrivare a una soluzione efficace e generalizzabile.



IL PENSIERO COMPUTAZIONALE

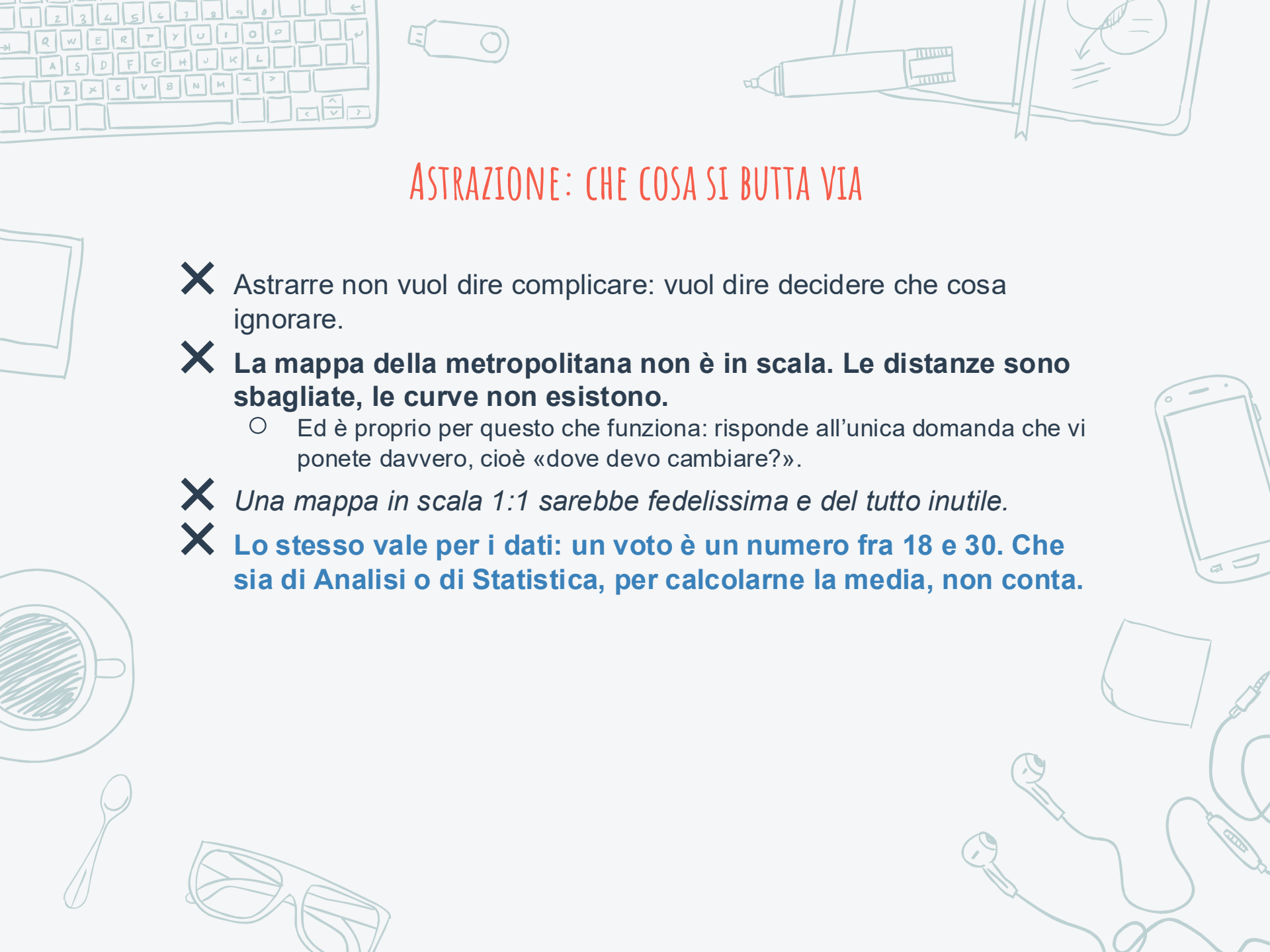
✗ Si può riassumere in alcune abilità fondamentali:

- **Decomposizione** → dividere un problema complesso in sottoproblemi più gestibili.
- **Riconoscimento di schemi** → individuare somiglianze o strutture ricorrenti tra problemi diversi.
- **Astrazione** → concentrarsi sugli elementi essenziali, ignorando i dettagli superflui.
- **Algoritmi** → progettare sequenze di passi chiari e ordinati per risolvere il problema.
- **Valutazione** → riflettere sull'efficacia e l'efficienza della soluzione, migliorandola se necessario.



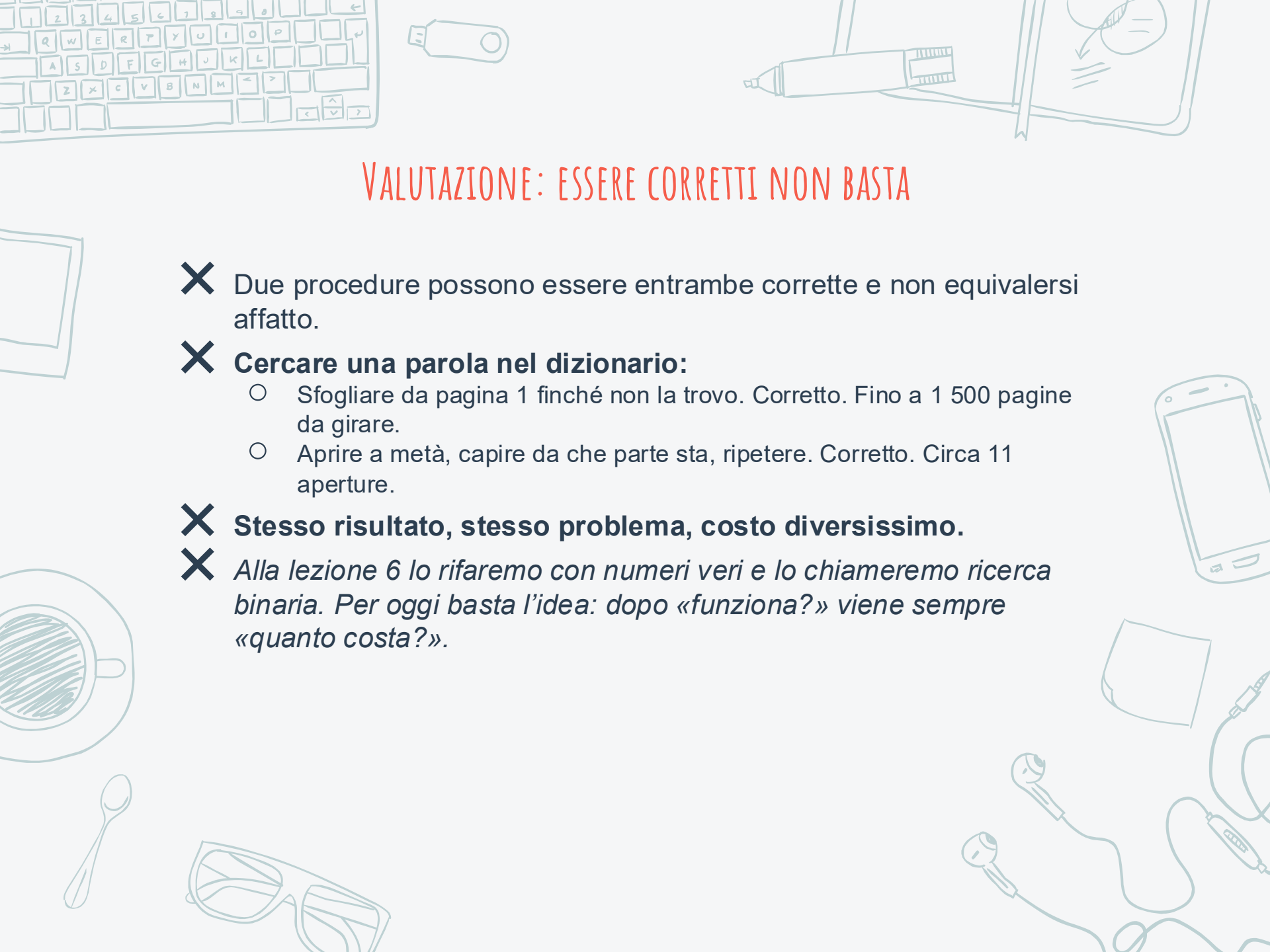
DECOMPOSIZIONE: DAL PROBLEMA AI SOTTOPROBLEMI

- ✗ Nessuno risolve un problema grande in un colpo solo. Lo si spezza finché i pezzi non sono ovvi.
- ✗ **Si procede per raffinamenti successivi:**
 - «Organizzare la festa» è troppo grande per essere eseguito.
 - Diventa: invitare, comprare, cucinare, sistemare la casa.
 - «Cucinare» diventa: scegliere il menu, fare la spesa, preparare.
 - Ci si ferma quando un passo si sa già fare senza pensarci.
- ✗ **La regola pratica: se non sai come si esegue, non è ancora un passo. È un sottoproblema.**



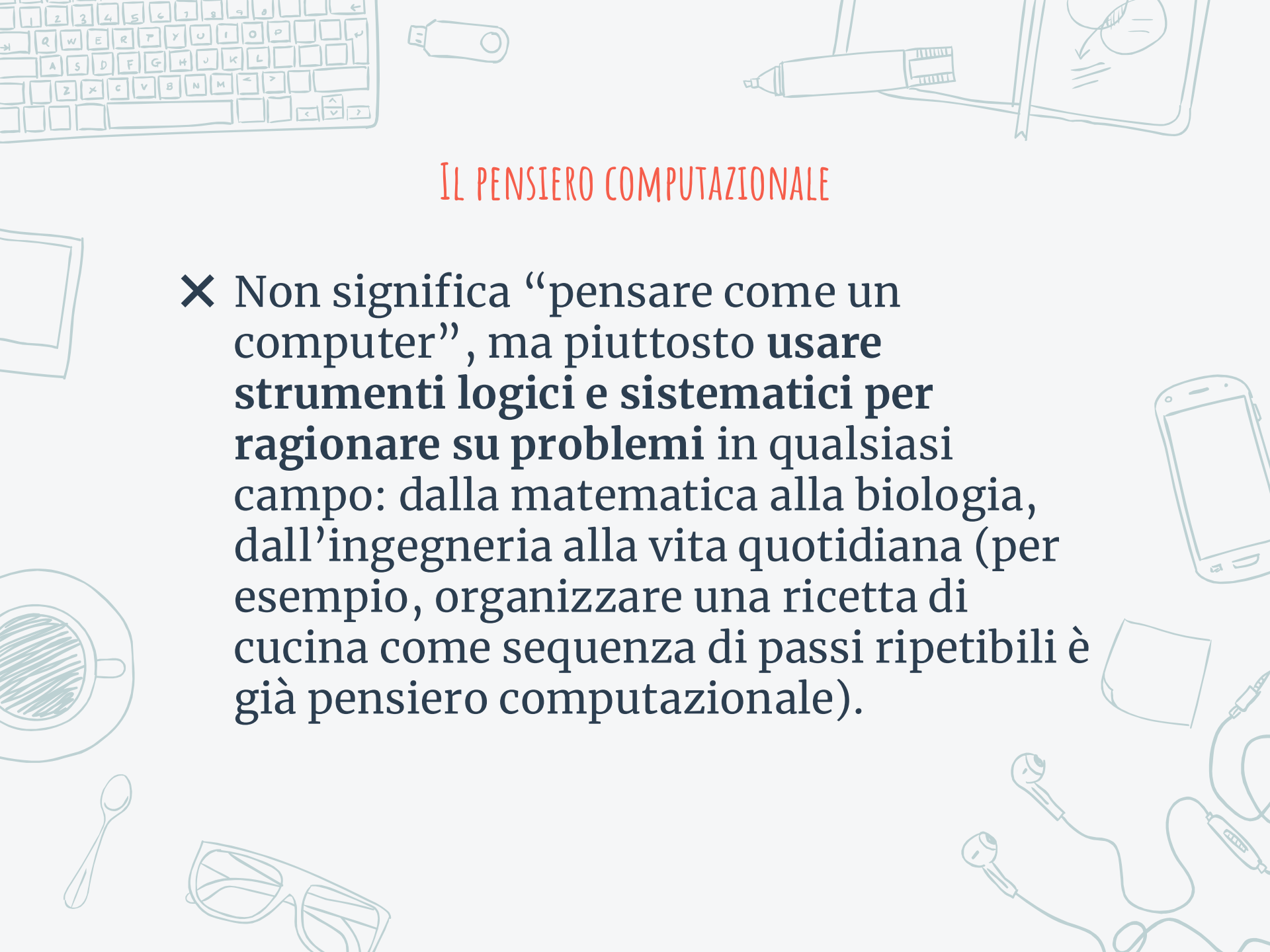
ASTRAZIONE: CHE COSA SI BUTTA VIA

- ✗ Astrarre non vuol dire complicare: vuol dire decidere che cosa ignorare.
- ✗ **La mappa della metropolitana non è in scala. Le distanze sono sbagliate, le curve non esistono.**
 - Ed è proprio per questo che funziona: risponde all'unica domanda che vi ponete davvero, cioè «dove devo cambiare?».
- ✗ *Una mappa in scala 1:1 sarebbe fedelissima e del tutto inutile.*
- ✗ **Lo stesso vale per i dati: un voto è un numero fra 18 e 30. Che sia di Analisi o di Statistica, per calcolarne la media, non conta.**



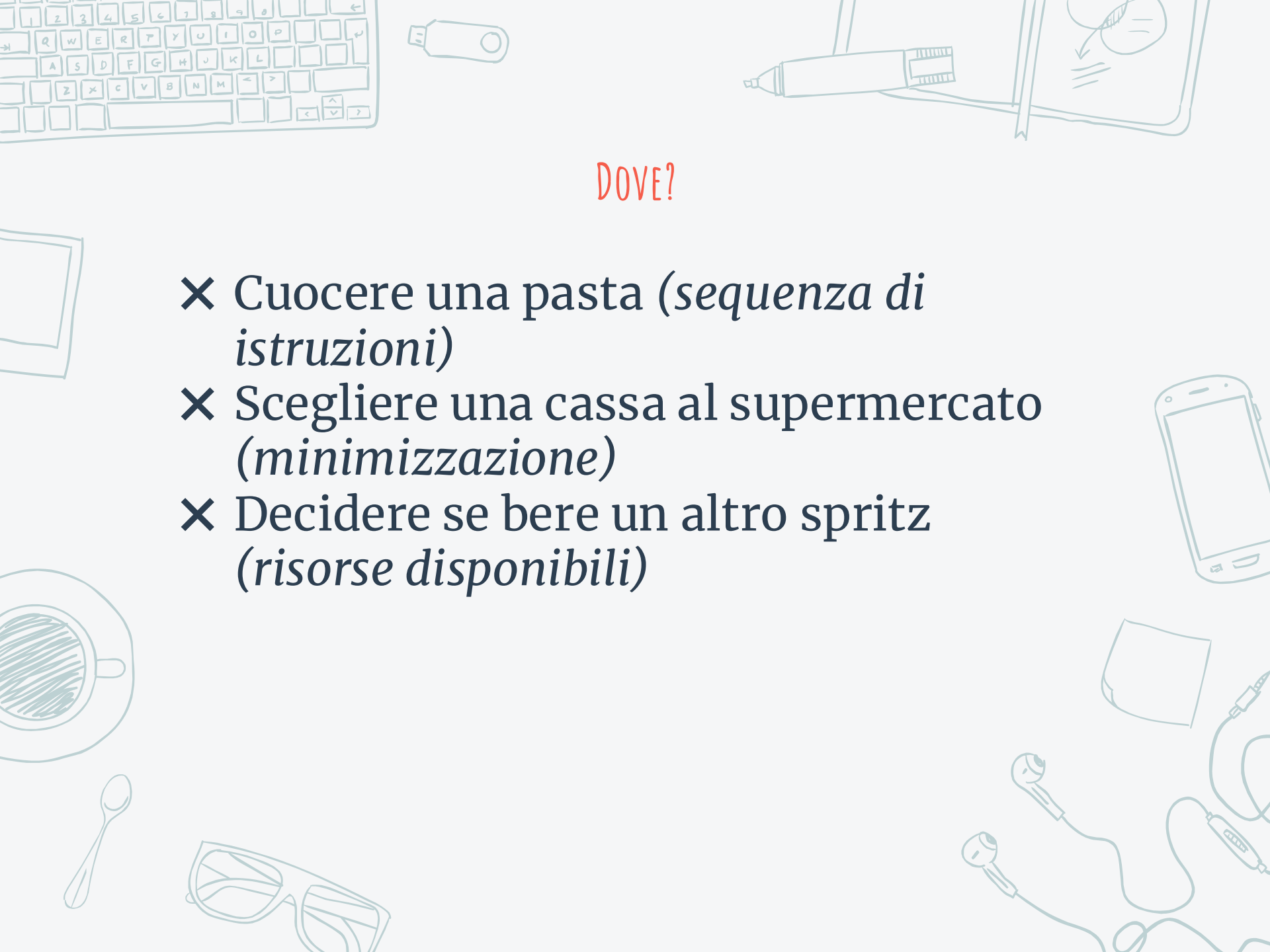
VALUTAZIONE: ESSERE CORRETTI NON BASTA

- ✗ Due procedure possono essere entrambe corrette e non equivalersi affatto.
- ✗ **Cercare una parola nel dizionario:**
 - Sfogliare da pagina 1 finché non la trovo. Corretto. Fino a 1 500 pagine da girare.
 - Aprire a metà, capire da che parte sta, ripetere. Corretto. Circa 11 aperture.
- ✗ **Stesso risultato, stesso problema, costo diversissimo.**
- ✗ *Alla lezione 6 lo rifaremo con numeri veri e lo chiameremo ricerca binaria. Per oggi basta l'idea: dopo «funziona?» viene sempre «quanto costa?».*



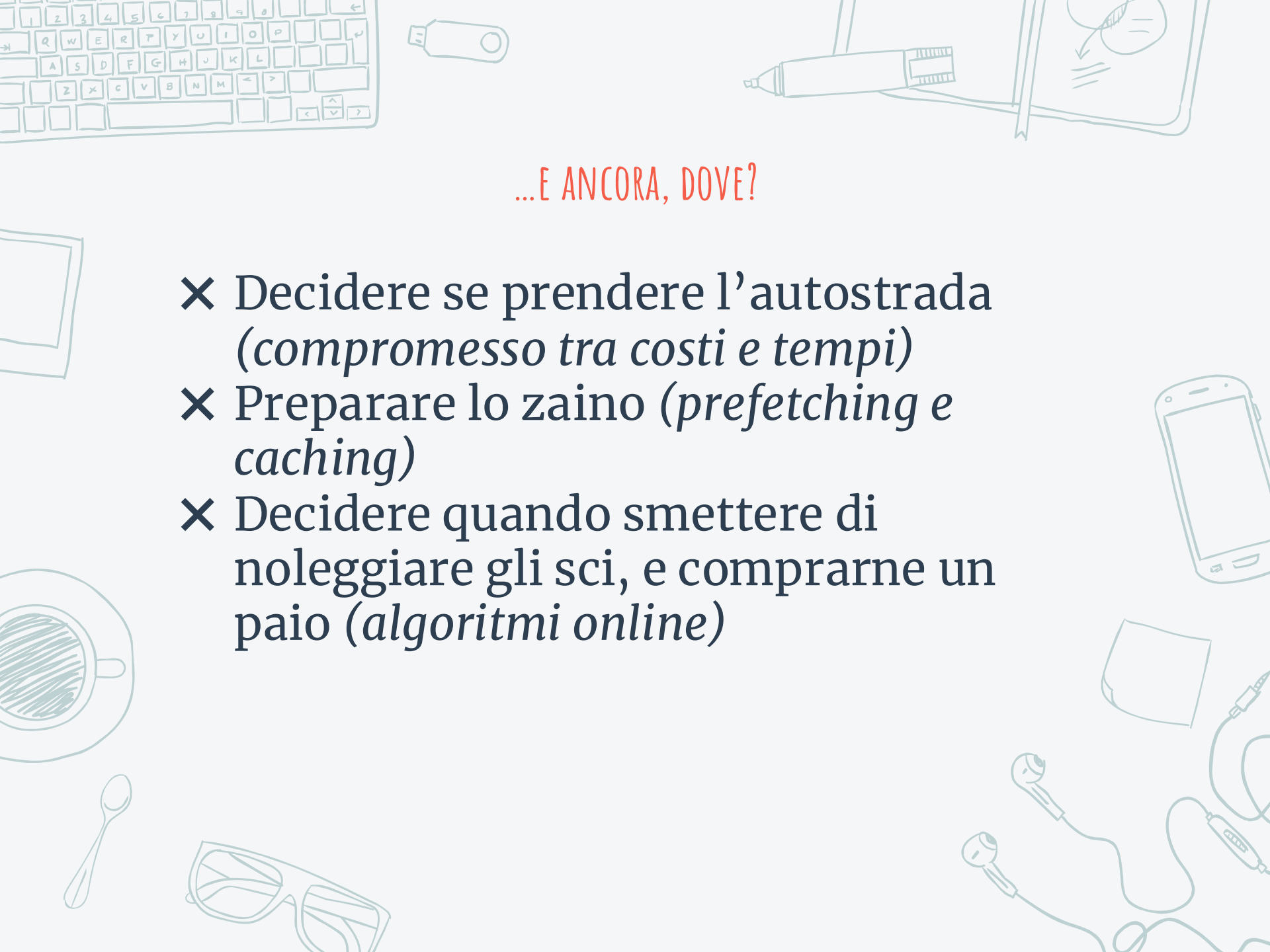
IL PENSIERO COMPUTAZIONALE

✗ Non significa “pensare come un computer”, ma piuttosto **usare strumenti logici e sistematici per ragionare su problemi** in qualsiasi campo: dalla matematica alla biologia, dall’ingegneria alla vita quotidiana (per esempio, organizzare una ricetta di cucina come sequenza di passi ripetibili è già pensiero computazionale).



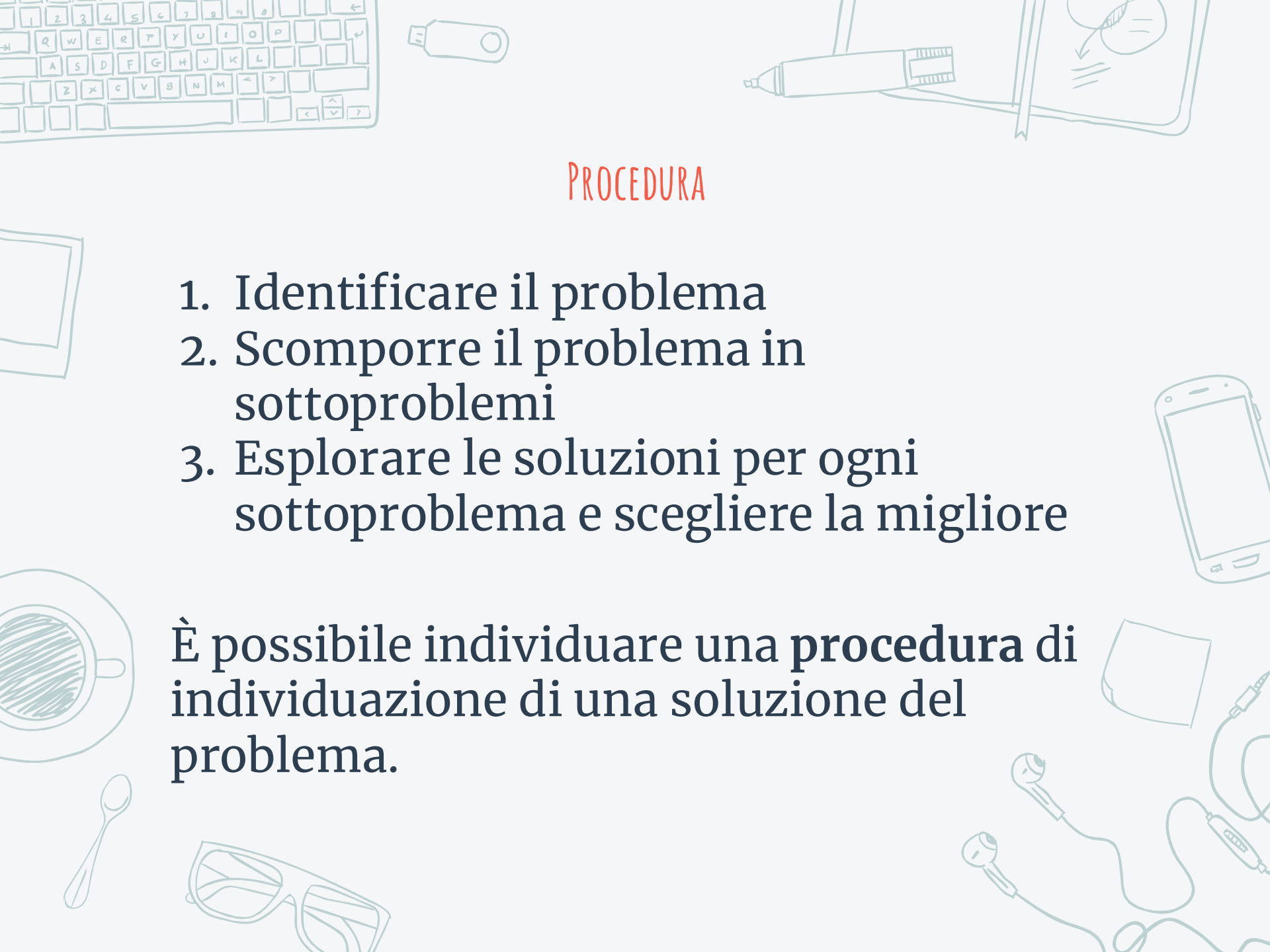
DOVE?

- ✗ Cuocere una pasta (*sequenza di istruzioni*)
- ✗ Scegliere una cassa al supermercato (*minimizzazione*)
- ✗ Decidere se bere un altro spritz (*risorse disponibili*)



...E ANCORA, DOVE?

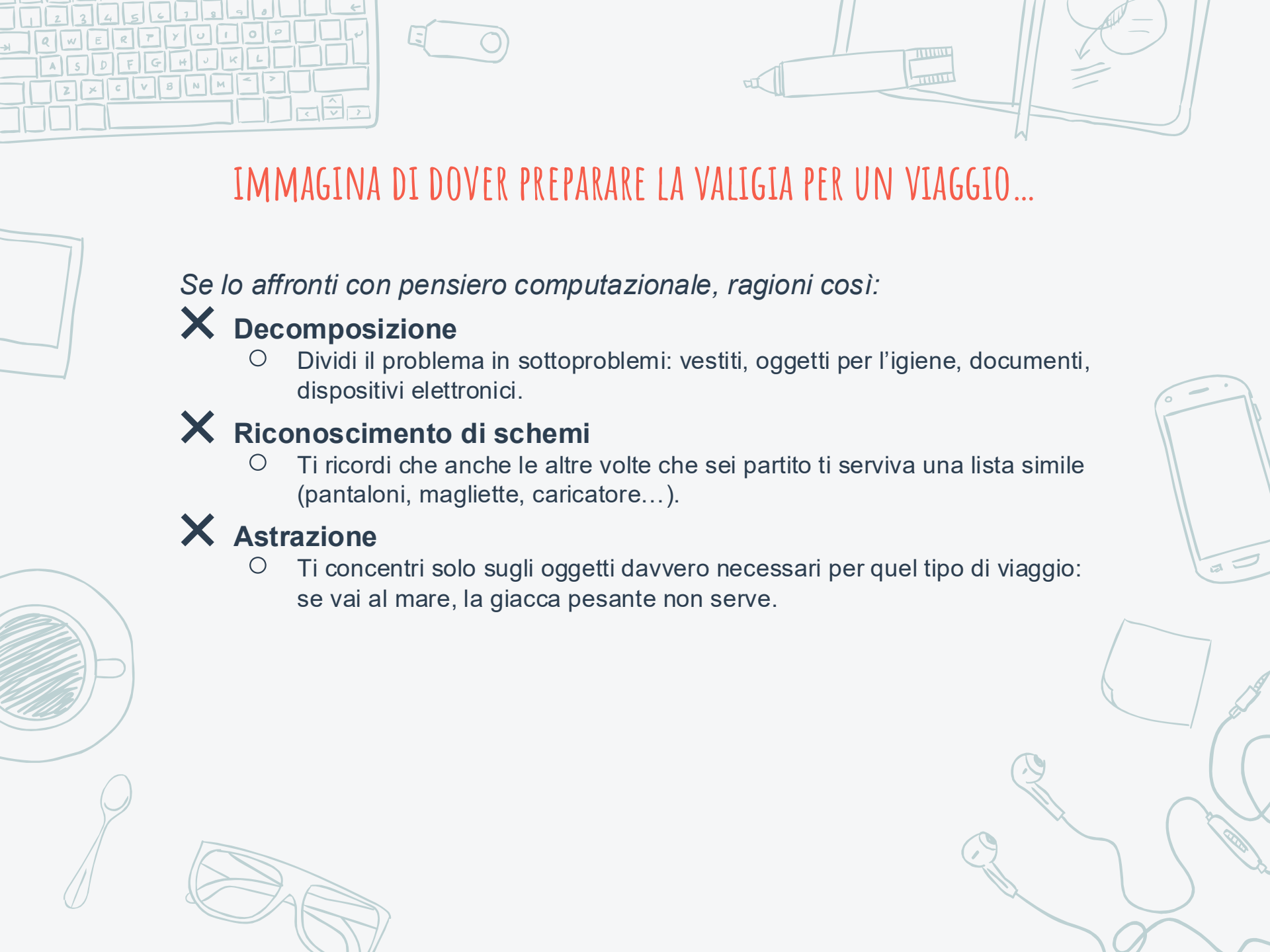
- ✗ Decidere se prendere l'autostrada (*compromesso tra costi e tempi*)
- ✗ Preparare lo zaino (*prefetching e caching*)
- ✗ Decidere quando smettere di noleggiare gli sci, e comprarne un paio (*algoritmi online*)



PROCEDURA

1. Identificare il problema
2. Scomporre il problema in sottoproblemi
3. Esplorare le soluzioni per ogni sottoproblema e scegliere la migliore

È possibile individuare una **procedura** di individuazione di una soluzione del problema.



IMMAGINA DI DOVER PREPARARE LA VALIGIA PER UN VIAGGIO...

Se lo affronti con pensiero computazionale, ragioni così:

✗ **Decomposizione**

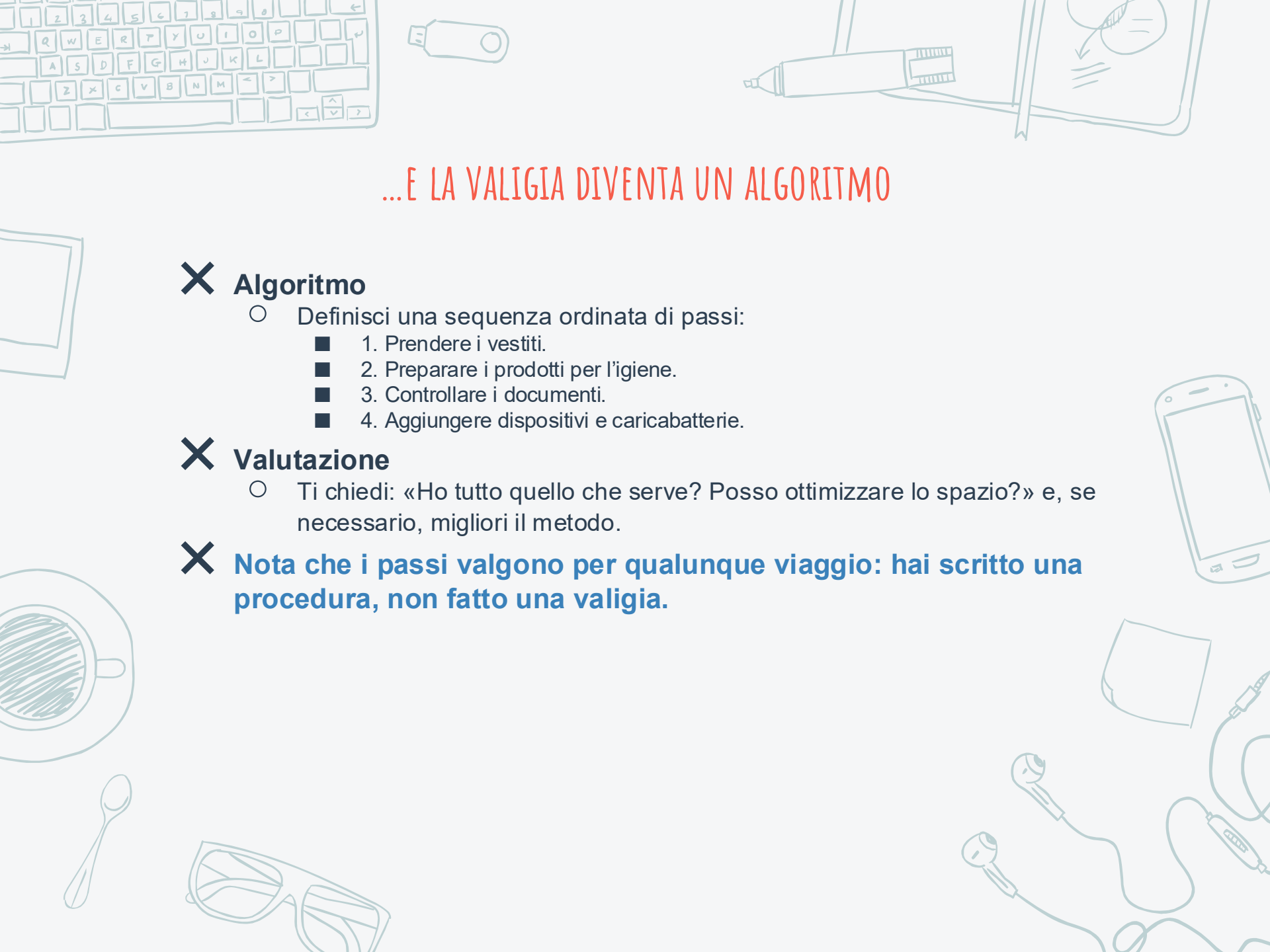
- Dividi il problema in sottoproblemi: vestiti, oggetti per l'igiene, documenti, dispositivi elettronici.

✗ **Riconoscimento di schemi**

- Ti ricordi che anche le altre volte che sei partito ti serviva una lista simile (pantaloni, magliette, caricatore...).

✗ **Astrazione**

- Ti concentri solo sugli oggetti davvero necessari per quel tipo di viaggio: se vai al mare, la giacca pesante non serve.



...E LA VALIGIA DIVENTA UN ALGORITMO

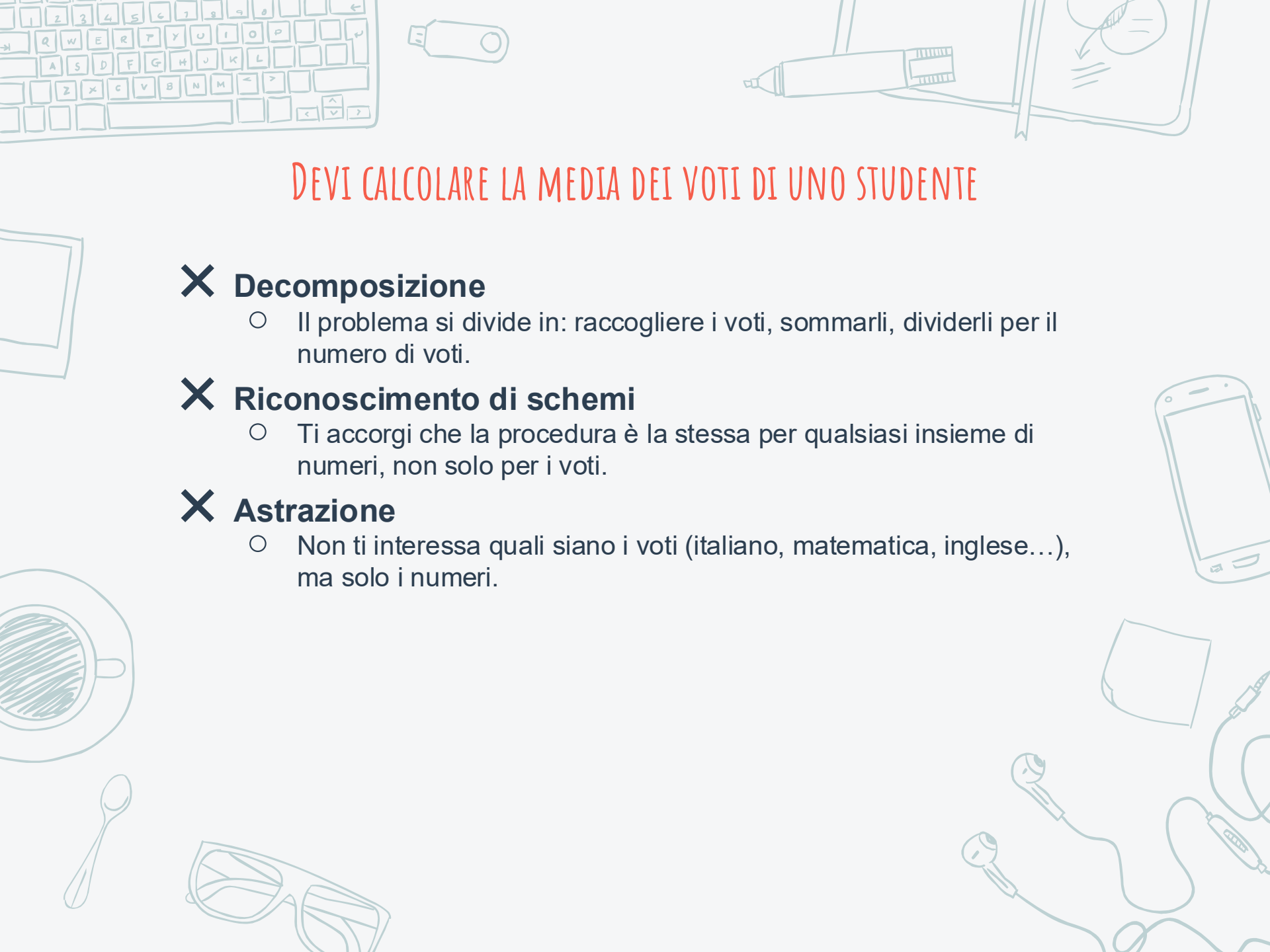
✗ **Algoritmo**

- Definisci una sequenza ordinata di passi:
 - 1. Prendere i vestiti.
 - 2. Preparare i prodotti per l'igiene.
 - 3. Controllare i documenti.
 - 4. Aggiungere dispositivi e caricabatterie.

✗ **Valutazione**

- Ti chiedi: «Ho tutto quello che serve? Posso ottimizzare lo spazio?» e, se necessario, migliori il metodo.

✗ **Nota che i passi valgono per qualunque viaggio: hai scritto una procedura, non fatto una valigia.**



DEVI CALCOLARE LA MEDIA DEI VOTI DI UNO STUDENTE

✗ **Decomposizione**

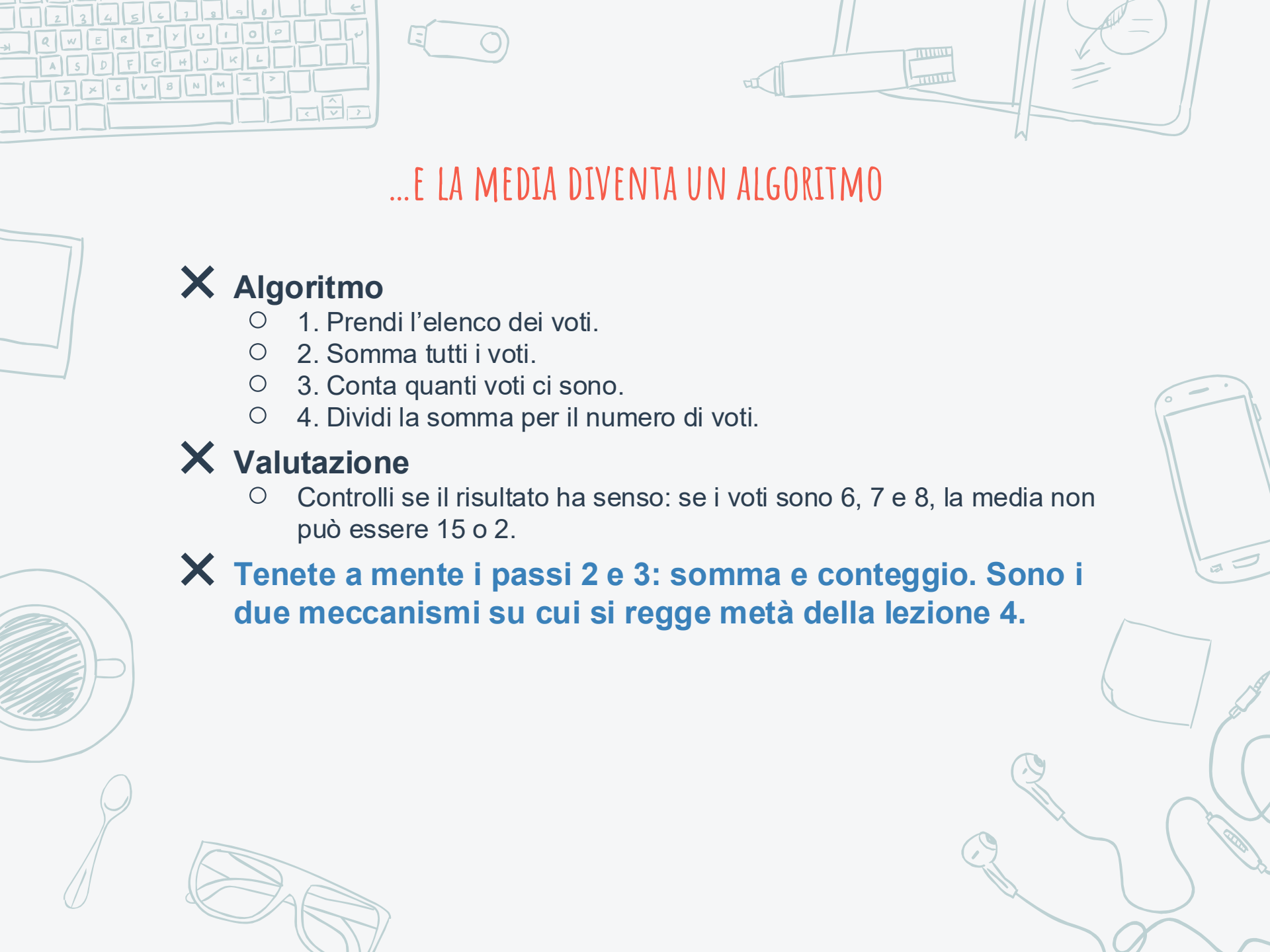
- Il problema si divide in: raccogliere i voti, sommarli, dividerli per il numero di voti.

✗ **Riconoscimento di schemi**

- Ti accorgi che la procedura è la stessa per qualsiasi insieme di numeri, non solo per i voti.

✗ **Astrazione**

- Non ti interessa quali siano i voti (italiano, matematica, inglese...), ma solo i numeri.



...E LA MEDIA DIVENTA UN ALGORITMO

✗ **Algoritmo**

- 1. Prendi l'elenco dei voti.
- 2. Somma tutti i voti.
- 3. Conta quanti voti ci sono.
- 4. Dividi la somma per il numero di voti.

✗ **Valutazione**

- Controlli se il risultato ha senso: se i voti sono 6, 7 e 8, la media non può essere 15 o 2.

✗ **Tenete a mente i passi 2 e 3: somma e conteggio. Sono i due meccanismi su cui si regge metà della lezione 4.**

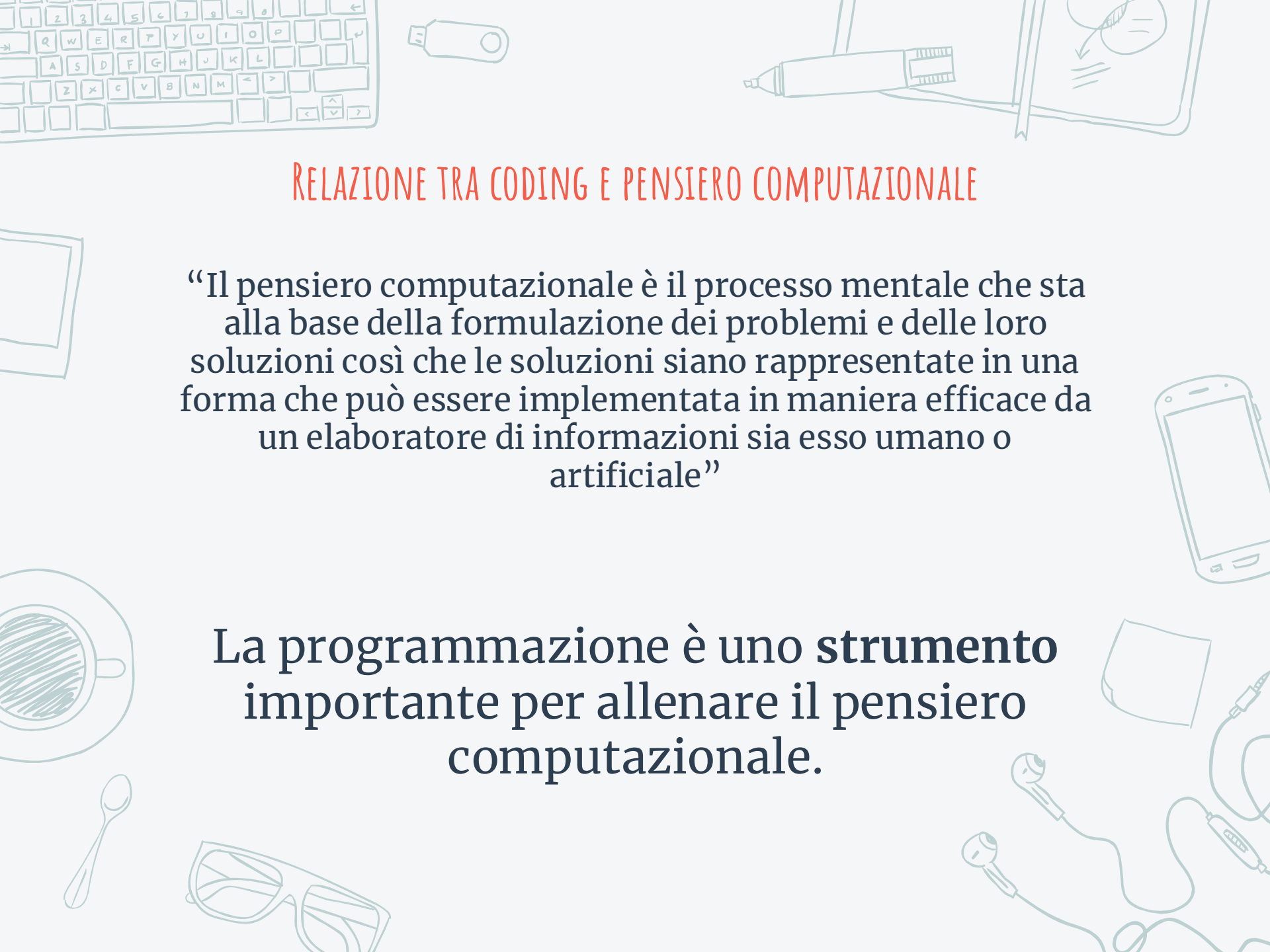


PROGRAMMAZIONE



PENSIERO COMPUTAZIONALE



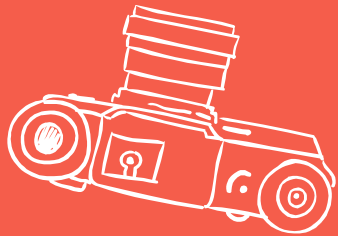


RELAZIONE TRA CODING E PENSIERO COMPUTAZIONALE

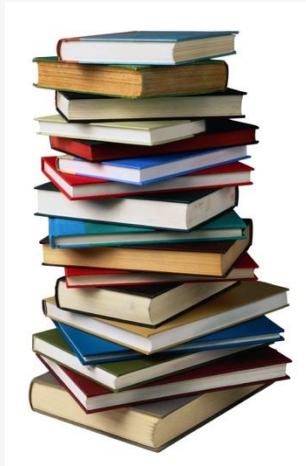
“Il pensiero computazionale è il processo mentale che sta alla base della formulazione dei problemi e delle loro soluzioni così che le soluzioni siano rappresentate in una forma che può essere implementata in maniera efficace da un elaboratore di informazioni sia esso umano o artificiale”

La programmazione è uno **strumento** importante per allenare il pensiero computazionale.

L'INFORMAZIONE DIGITALE



INFORMAZIONE DIGITALE: TUTTO DIVENTA <<BIT>>



Testo, Musica, Video,
Immagini: tutto viene
trasformato in una sequenza
di 0 e 1

01100100



PERCHÉ DUE STATI, E NON DIECI

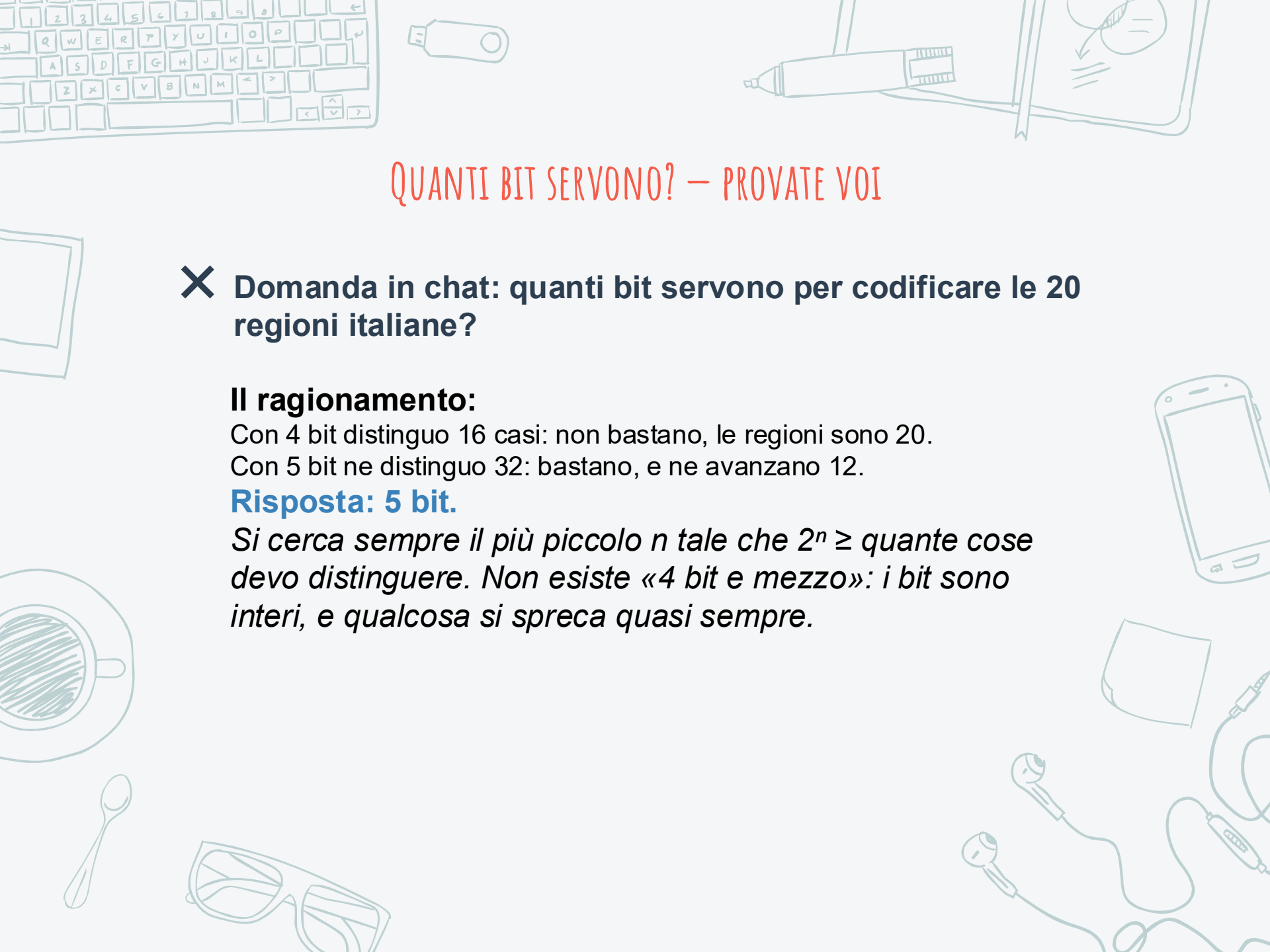
- ✗ *La domanda giusta non è «perché il binario?», ma «perché non il decimale?».*
- ✗ **Dentro un calcolatore l'informazione è una tensione elettrica.**
 - Con dieci livelli, ognuno disterebbe dal vicino tre decimi di volt: un po' di calore, un cavo lungo, e il valore si sporca.
 - Con due livelli ben distanti — c'è corrente / non c'è corrente — servirebbe un disturbo enorme per confondere l'uno con lo zero.
- ✗ **Il binario non è una scelta matematica elegante: è una scelta ingegneristica. È il modo più economico di essere affidabili.**

UN BIT È UNA RISPOSTA SÌ / NO

Con un bit distinguo 2 casi. Con n bit ne distinguo 2^n : ogni bit in più raddoppia.

Bit	Casi distinguibili	Che cosa ci sta dentro
1	2	acceso / spento
2	4	i semi delle carte
4	16	una cifra esadecimale
8	256	un carattere ASCII esteso — è un byte
16	65 536	l'altezza di un suono campionato

Otto bit = un byte. È l'unità con cui si misura tutto il resto.



QUANTI BIT SERVONO? — PROVATE VOI

✗ **Domanda in chat: quanti bit servono per codificare le 20 regioni italiane?**

Il ragionamento:

Con 4 bit distinguo 16 casi: non bastano, le regioni sono 20.

Con 5 bit ne distinguo 32: bastano, e ne avanzano 12.

Risposta: 5 bit.

Si cerca sempre il più piccolo n tale che $2^n \geq$ quante cose devo distinguere. Non esiste «4 bit e mezzo»: i bit sono interi, e qualcosa si spreca quasi sempre.

DA BINARIO A DECIMALE

✗ Ogni posizione vale una potenza di 2. Si sommano solo le posizioni che valgono 1.

✗ Prendiamo il numero già stampato due slide fa: 01100100

○	128	64	32	16	8	4	2	1	← i pesi
○	0	1	1	0	0	1	0	0	← le

cifre

○ $64 + 32 + 4 = 100$

✗ In decimale usate le potenze di 10 e non ci pensate: 407 è $4 \cdot 100 + 0 \cdot 10 + 7$. Qui è identico, con il 2 al posto del 10.

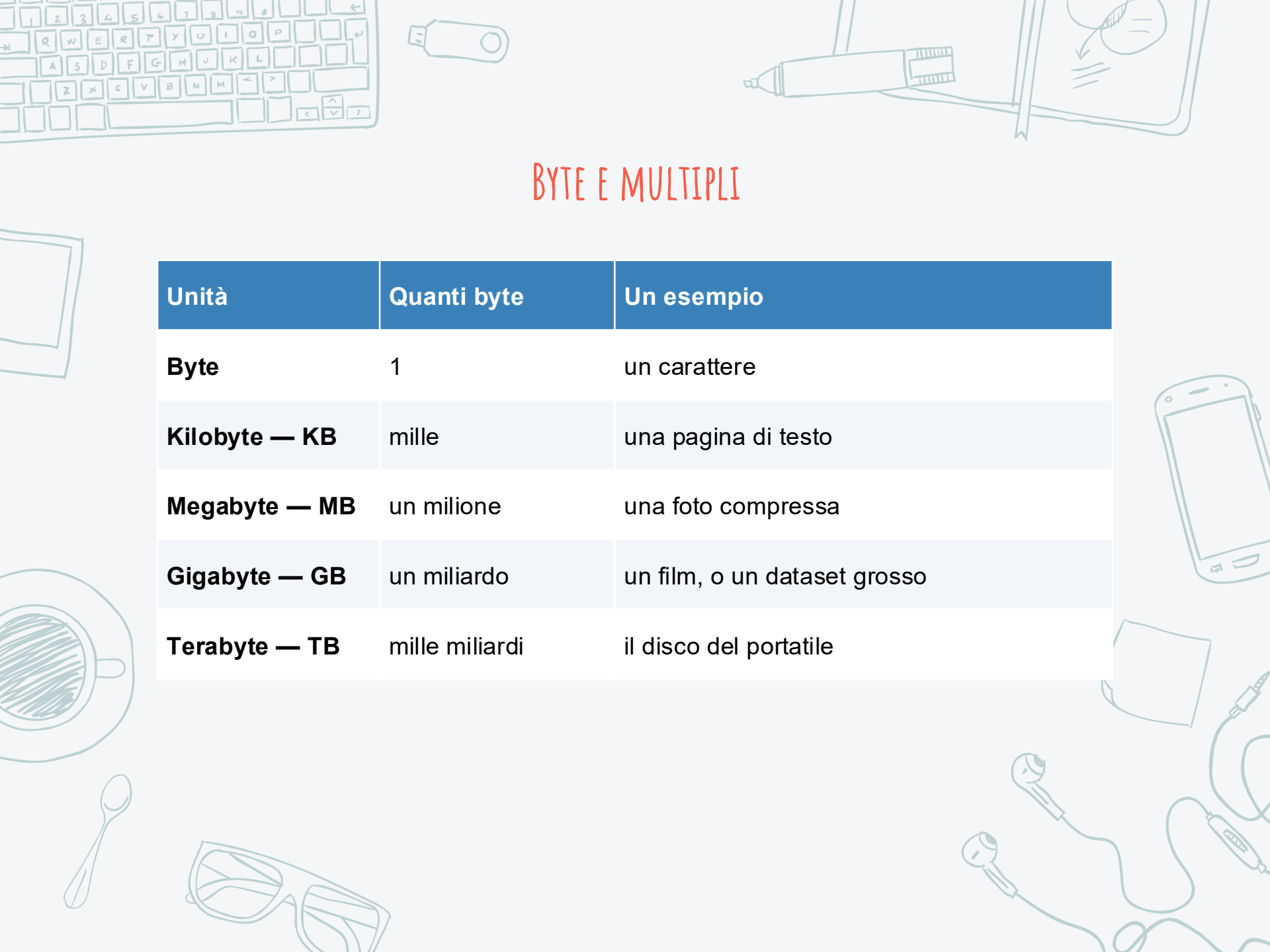
DA DECIMALE A BINARIO

✗ Si divide per 2 ripetutamente e si tengono i resti. Poi si leggono dal basso verso l'alto.

✗ *Convertiamo 100:*

○	100	: 2 = 50	resto 0
○	50	: 2 = 25	resto 0
○	25	: 2 = 12	resto 1
○	12	: 2 = 6	resto 0
○	6	: 2 = 3	resto 0
○	3	: 2 = 1	resto 1
○	1	: 2 = 0	resto 1

✗ **Resti letti a ritroso: 1100100 → con otto bit, 01100100.**
Torna.



BYTE E MULTIPLI

Unità	Quanti byte	Un esempio
Byte	1	un carattere
Kilobyte — KB	mille	una pagina di testo
Megabyte — MB	un milione	una foto compressa
Gigabyte — GB	un miliardo	un film, o un dataset grosso
Terabyte — TB	mille miliardi	il disco del portatile

PERCHÉ LA CHIAVETTA DA 64 GB NE MOSTRA 59,6

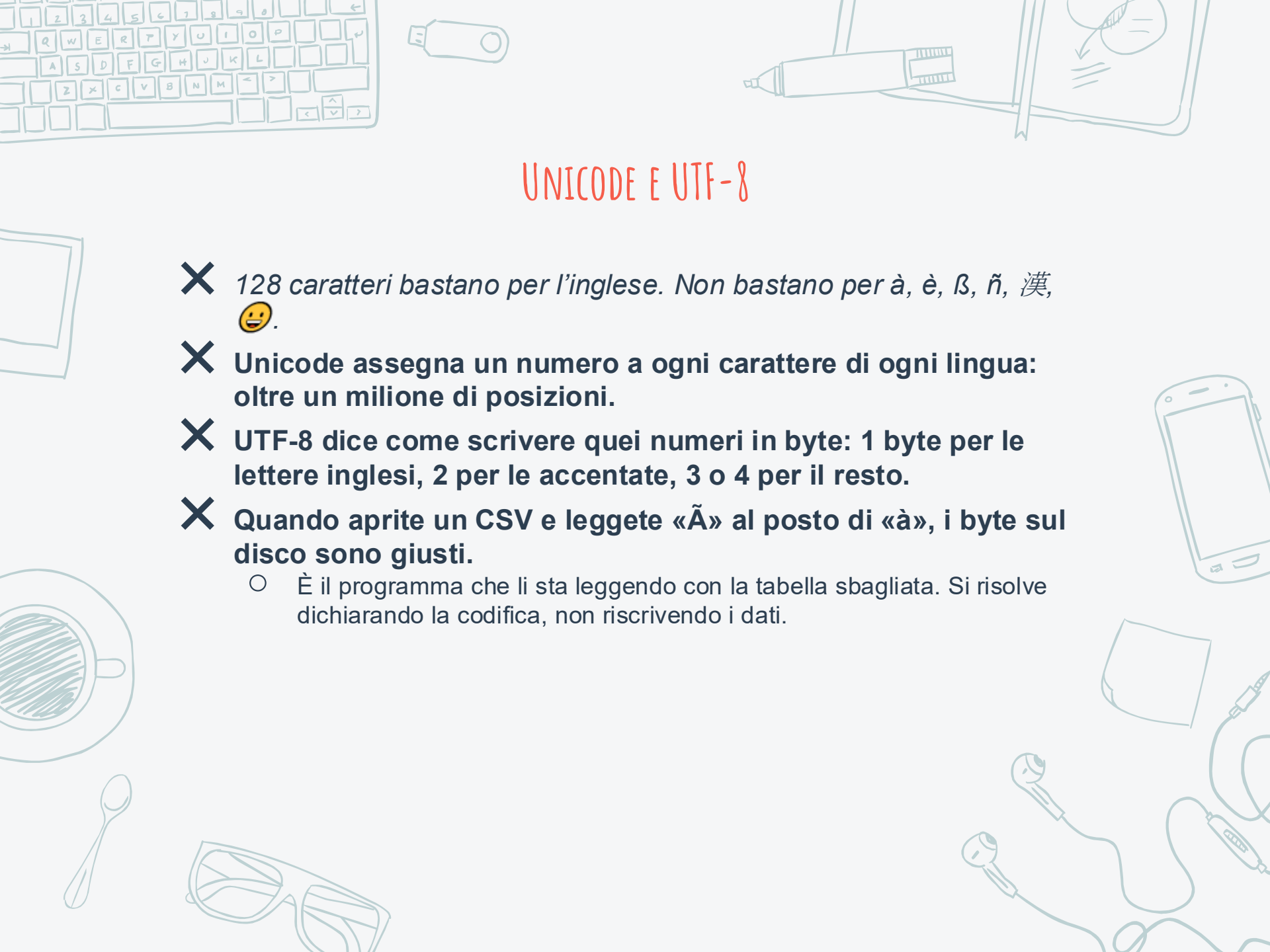
- ✗ *Non vi hanno truffati: state usando due definizioni diverse della stessa parola.*
- ✗ **Il produttore conta in potenze di 10: 1 GB = 1 000 000 000 byte.**
 - 64 GB commerciali = 64 000 000 000 byte.
- ✗ **Il sistema operativo conta in potenze di 2: 1 GiB = 2^{30} = 1 073 741 824 byte.**
 - $64\,000\,000\,000 : 1\,073\,741\,824 \approx 59,6$

E IL TESTO? UNA TABELLA, E NIENT'ALTRO

ASCII, 1963: sette bit, 128 caratteri. Ogni lettera è un numero deciso a tavolino.

Carattere	Codice	In binario, su 8 bit
spazio	32	00100000
0	48	00110000
A	65	01000001
a	97	01100001

Le maiuscole distano 32 dalle minuscole: è un solo bit di differenza. Non è un caso, è progettazione.



UNICODE E UTF-8

- ✗ *128 caratteri bastano per l'inglese. Non bastano per à, è, ß, ñ, 漢, 😊.*
- ✗ **Unicode assegna un numero a ogni carattere di ogni lingua: oltre un milione di posizioni.**
- ✗ **UTF-8 dice come scrivere quei numeri in byte: 1 byte per le lettere inglesi, 2 per le accentate, 3 o 4 per il resto.**
- ✗ **Quando aprite un CSV e leggete «Ã» al posto di «à», i byte sul disco sono giusti.**
 - È il programma che li sta leggendo con la tabella sbagliata. Si risolve dichiarando la codifica, non riscrivendo i dati.

UN'IMMAGINE È UNA GRIGLIA DI DOMANDE SÌ / NO

- ✗ **Una foto non è un disegno: è una tabella di numeri.**
- ✗ **In bianco e nero basta un bit per pixel: acceso o spento.**
 - Una griglia 36×36 è fatta di 1 296 pixel, cioè 1 296 bit: 162 byte in tutto.
- ✗ **A colori servono tre numeri per pixel — quanto rosso, quanto verde, quanto blu.**
 - Un byte per canale, cioè 256 livelli ciascuno: 3 byte per pixel, e 16,7 milioni di colori possibili.
- ✗ **Ingrandite qualunque immagine finché non vedete i quadretti: quello è il dato vero. Il resto è interpretazione dei vostri occhi.**

QUANTO PESA DAVVERO UNA FOTO

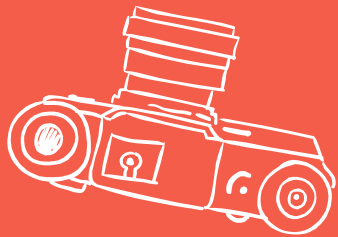
- ✗ *Il telefono dice «12 megapixel». Facciamo il conto.*
- ✗ **$12\,000\,000 \text{ pixel} \times 3 \text{ byte per pixel} = 36\,000\,000 \text{ byte} \approx 36 \text{ MB}$**
- ✗ **Ma sul telefono quella foto occupa 3 o 4 MB. Dove sono finiti gli altri trenta?**
 - Li ha tolti la compressione JPEG, sfruttando il fatto che i pixel vicini si somigliano e che il vostro occhio non nota la differenza.
 - È compressione con perdita: i dati buttati via non tornano più.
- ✗ *Quando comprimate un dataset, invece, la perdita non è ammessa.*

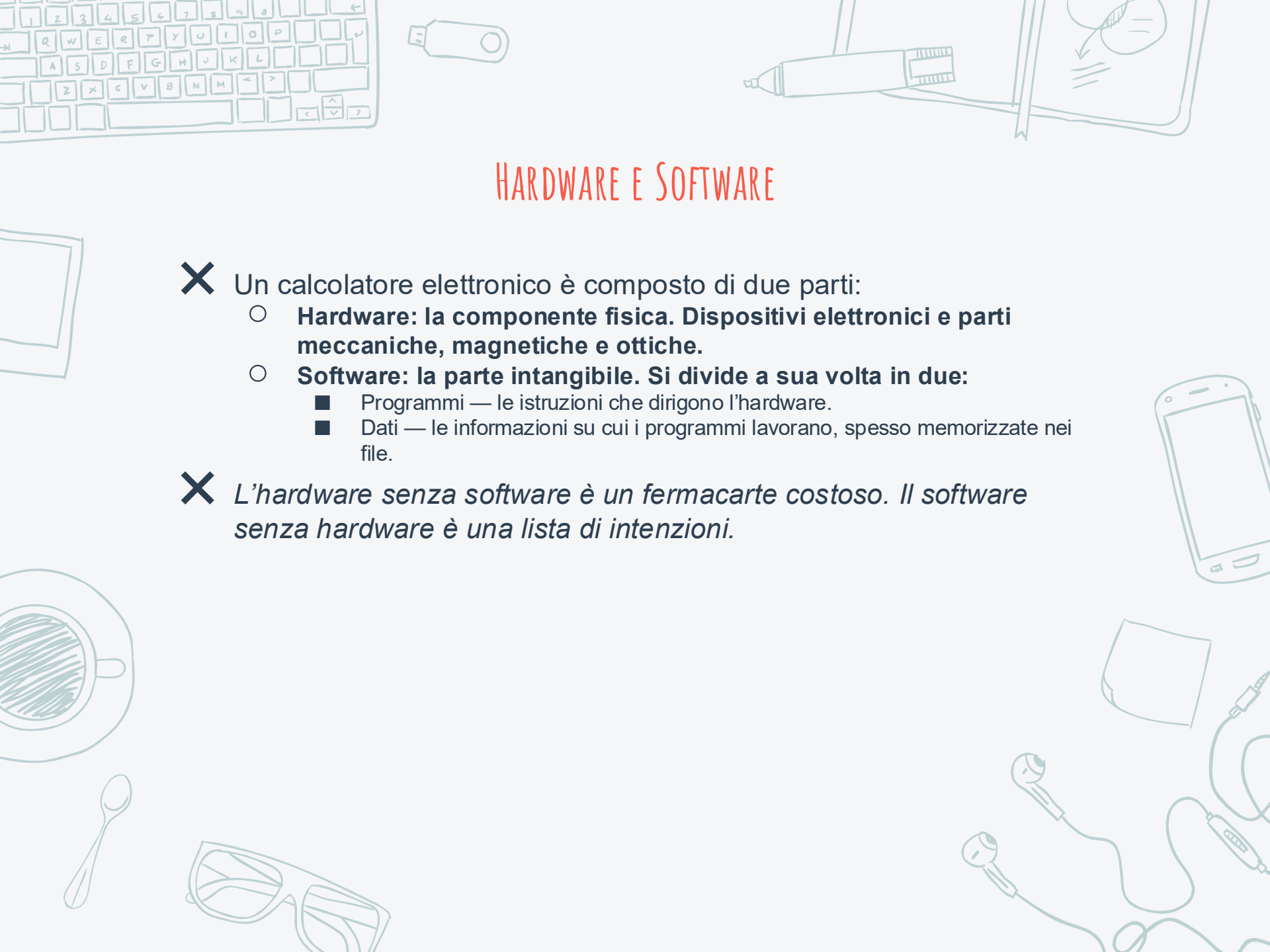


CAMPIONAMENTO

- ✗ *Un suono è un fenomeno continuo. Un file è una sequenza finita di numeri. Come si passa dall'uno all'altro?*
- ✗ **Campionamento — si misura a intervalli regolari.**
 - Un CD misura 44100 volte al secondo. Fra una misura e l'altra, nessuno sa che cosa è successo.
- ✗ **Quantizzazione — ogni misura si arrotonda a un valore disponibile.**
 - Con 16 bit ci sono 65 536 livelli. Il valore vero cade quasi sempre in mezzo a due, e va approssimato.

COM'È FATTA LA MACCHINA





HARDWARE E SOFTWARE

- ✗ Un calcolatore elettronico è composto di due parti:
 - **Hardware:** la componente fisica. Dispositivi elettronici e parti meccaniche, magnetiche e ottiche.
 - **Software:** la parte intangibile. Si divide a sua volta in due:
 - Programmi — le istruzioni che dirigono l'hardware.
 - Dati — le informazioni su cui i programmi lavorano, spesso memorizzate nei file.
- ✗ *L'hardware senza software è un fermacarte costoso. Il software senza hardware è una lista di intenzioni.*



IL MODELLO DI VON NEUMANN

1945: John von Neumann descrive un'architettura in cui programma e dati stanno nella stessa memoria. Ottant'anni dopo, il vostro portatile è ancora questo.

CPU

Unità di controllo — decide quale istruzione tocca
ALU — calcola e confronta

Memoria centrale — RAM

Contiene le istruzioni del programma in corso e i dati su cui lavora

BUS — il canale su cui viaggiano dati, indirizzi e comandi

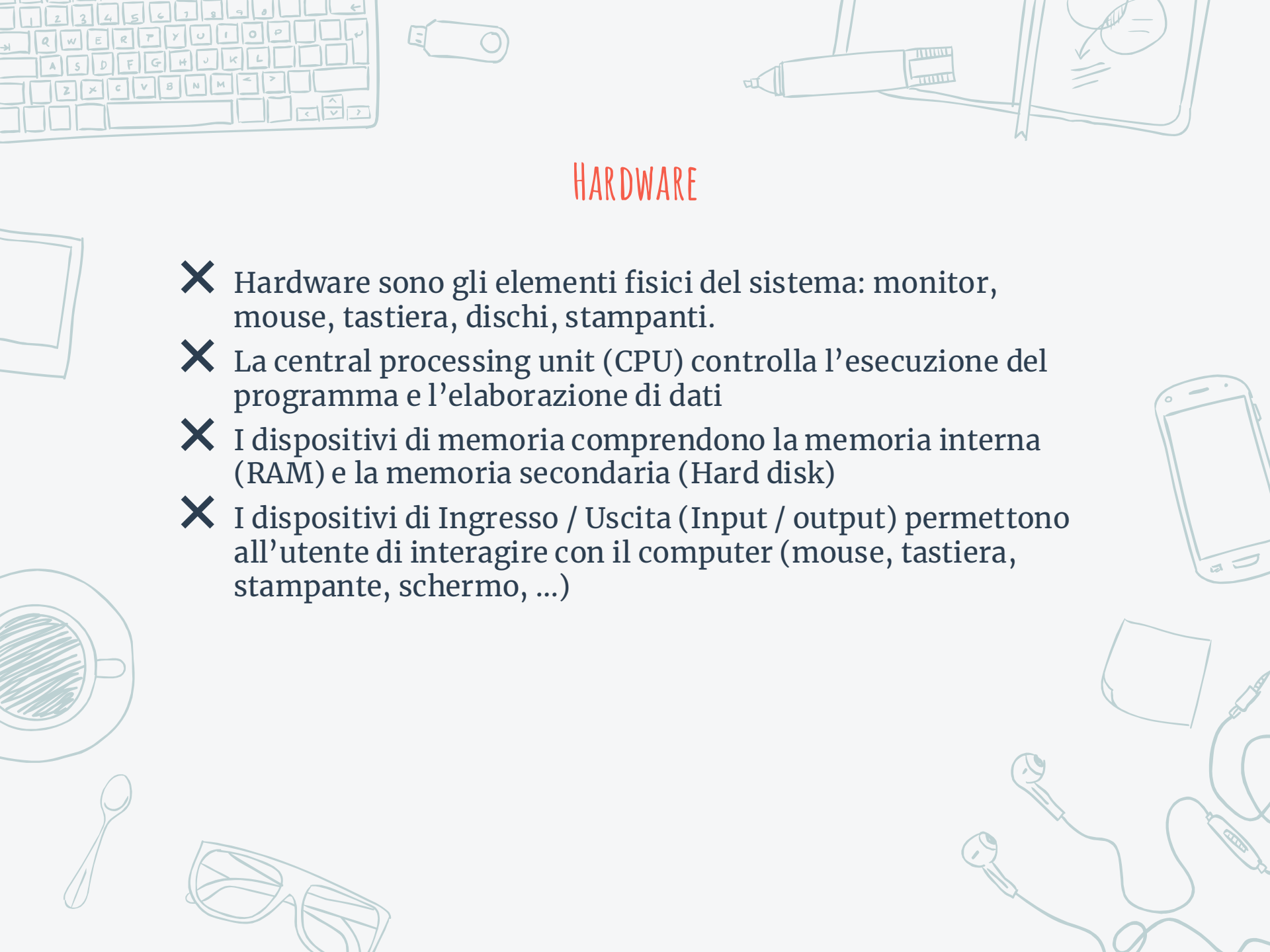
Memoria di massa

Disco o SSD: ciò che resta quando si spegne

Periferiche di ingresso/uscita

Tastiera, mouse, schermo, rete

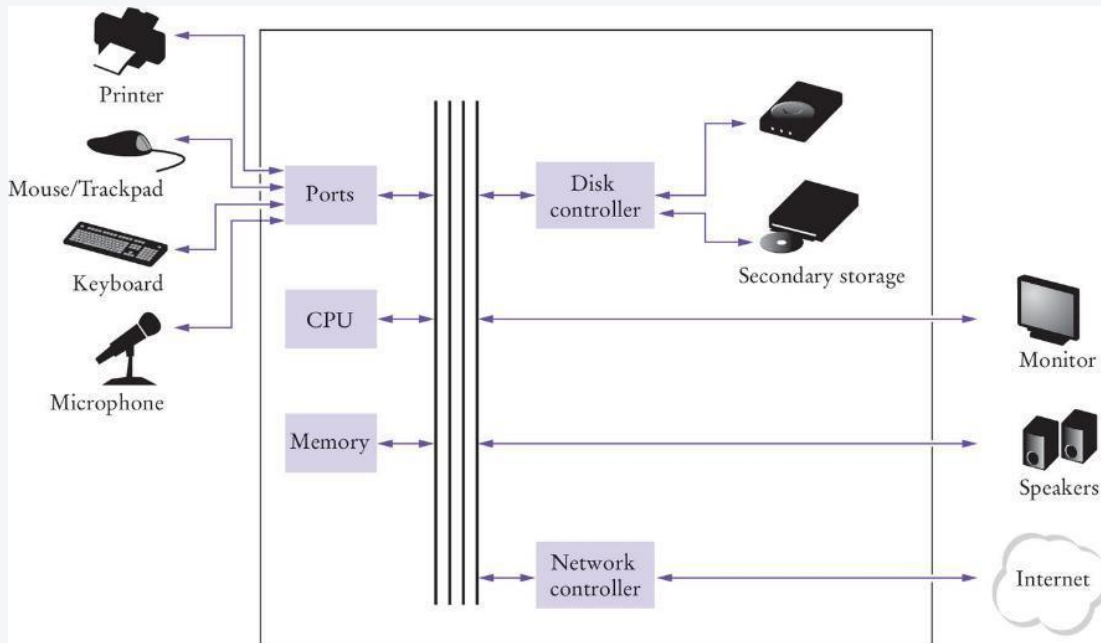
Un solo esecutore, una sola istruzione alla volta, in un ordine deciso. È per questo che un algoritmo deve essere una sequenza.



HARDWARE

- ✗ Hardware sono gli elementi fisici del sistema: monitor, mouse, tastiera, dischi, stampanti.
- ✗ La central processing unit (CPU) controlla l'esecuzione del programma e l'elaborazione di dati
- ✗ I dispositivi di memoria comprendono la memoria interna (RAM) e la memoria secondaria (Hard disk)
- ✗ I dispositivi di Ingresso / Uscita (Input / output) permettono all'utente di interagire con il computer (mouse, tastiera, stampante, schermo, ...)

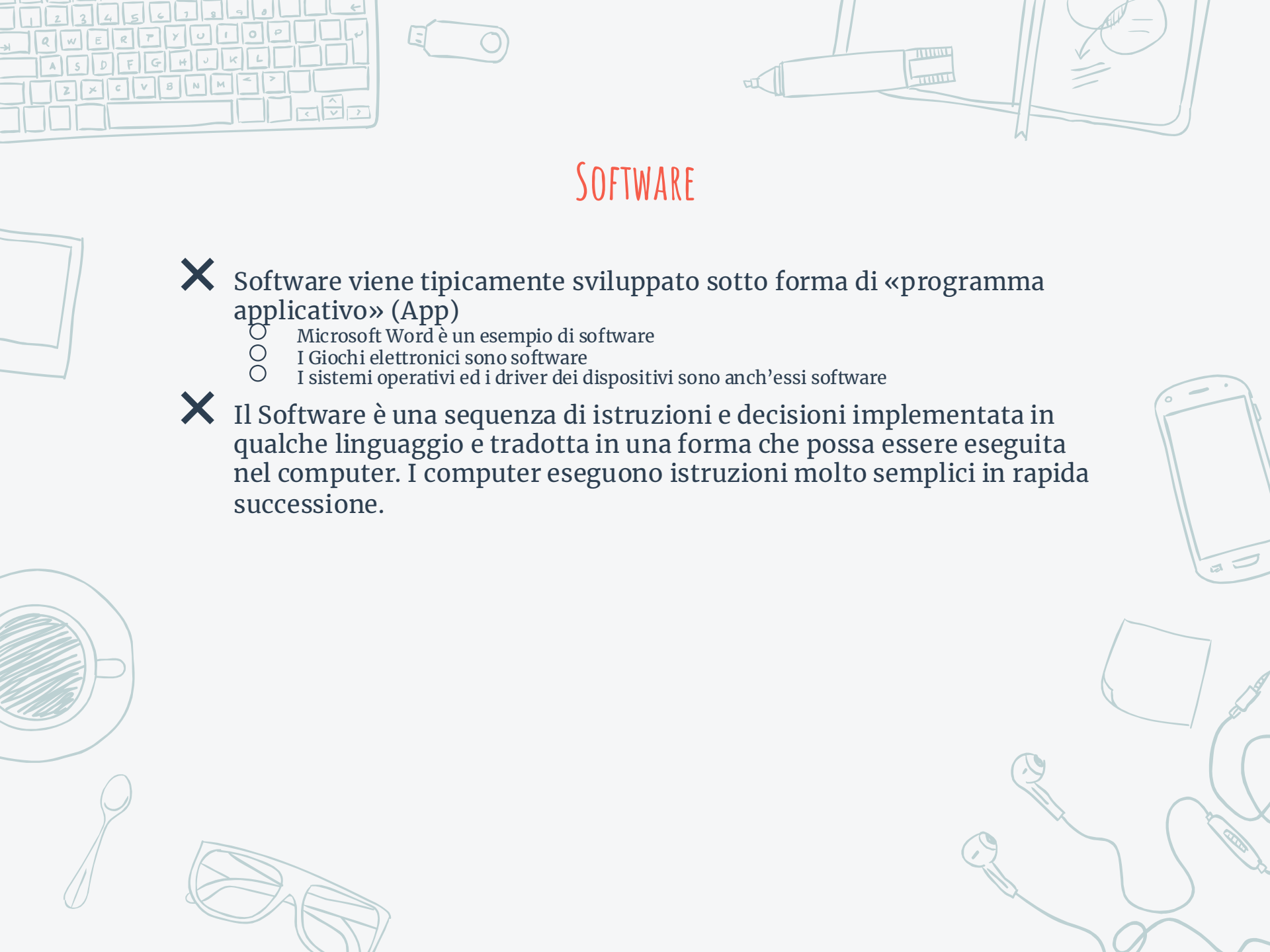
VISTA SEMPLIFICATA DELL'HARDWARE DI UN PC



DUE MEMORIE, DUE RUOLI

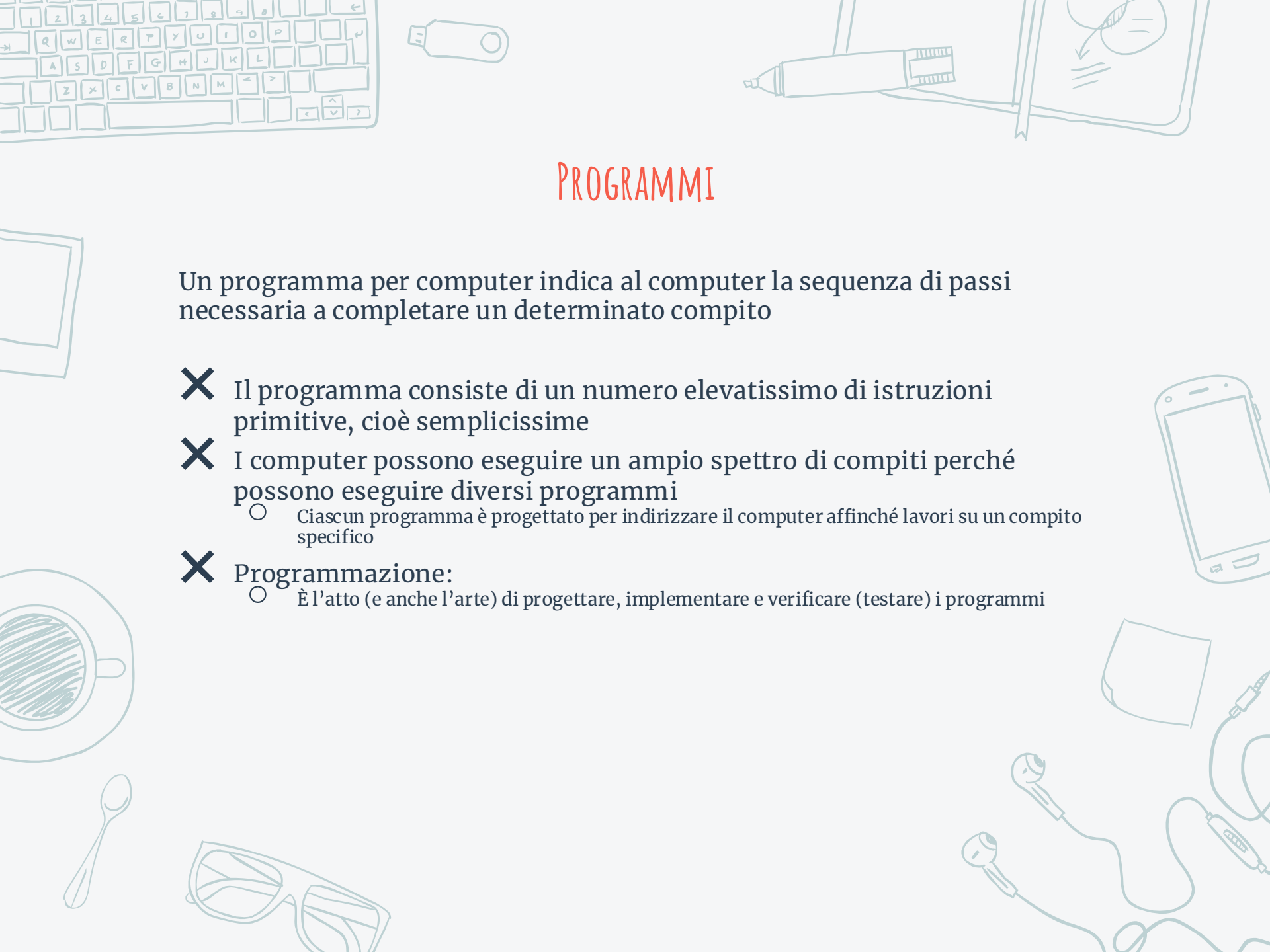
	Memoria centrale — RAM	Memoria di massa — disco / SSD
Velocità	Molto veloce	Lenta: da mille a un milione di volte
Persistenza	Volatile: si spegne, si svuota	Persistente: resta
Capacità	Poca — qualche GB	Molta — centinaia di GB o TB
A che serve	Ci sta il programma mentre gira	Ci stanno file, dataset, programmi spenti

Quando dalla prossima lezione diremo «variabile», intendete una casella con un nome dentro la RAM.



SOFTWARE

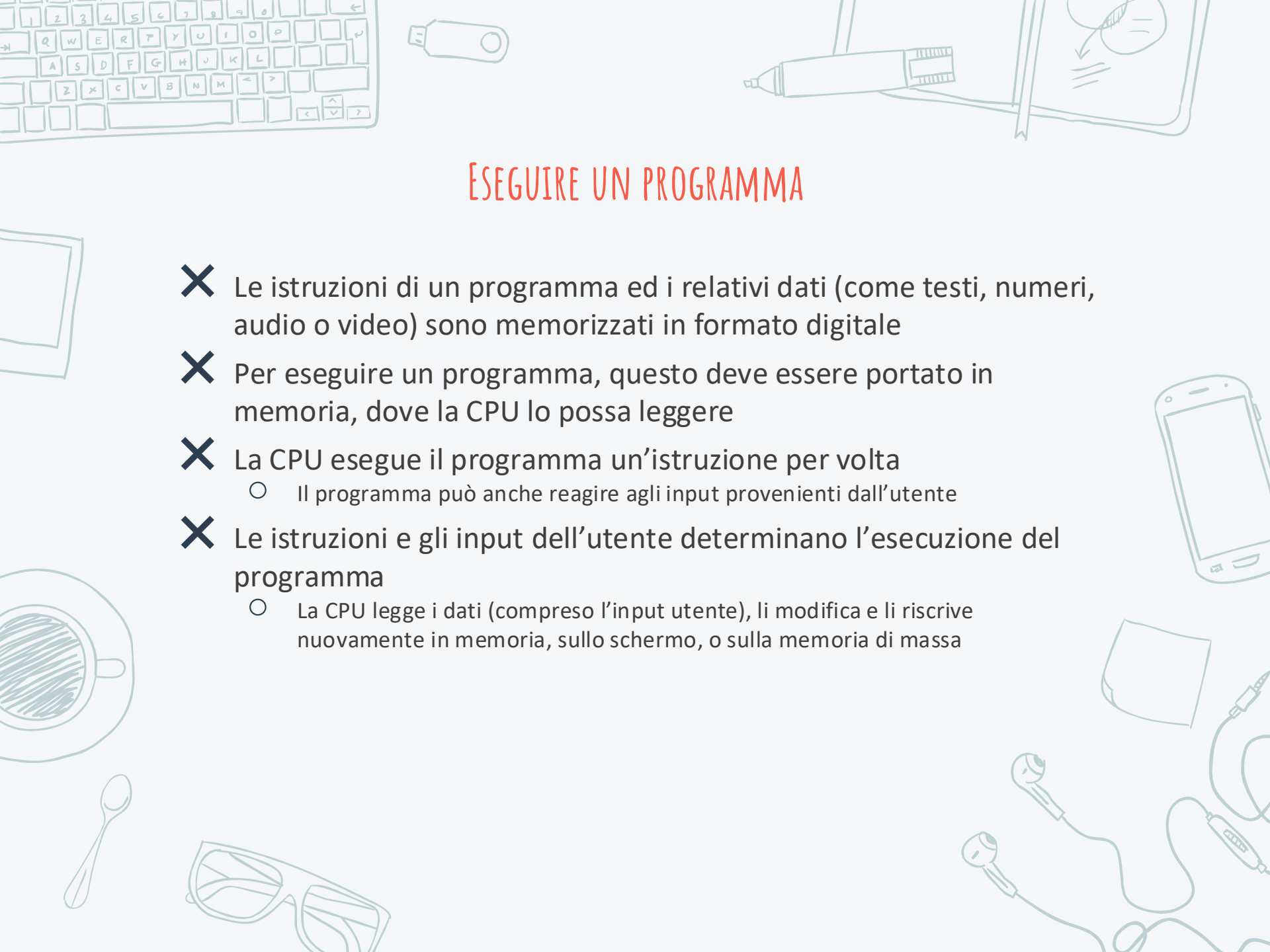
- ✗ Software viene tipicamente sviluppato sotto forma di «programma applicativo» (App)
 - Microsoft Word è un esempio di software
 - I Giochi elettronici sono software
 - I sistemi operativi ed i driver dei dispositivi sono anch'essi software
- ✗ Il Software è una sequenza di istruzioni e decisioni implementata in qualche linguaggio e tradotta in una forma che possa essere eseguita nel computer. I computer eseguono istruzioni molto semplici in rapida successione.



PROGRAMMI

Un programma per computer indica al computer la sequenza di passi necessaria a completare un determinato compito

- ✗ Il programma consiste di un numero elevatissimo di istruzioni primitive, cioè semplicissime
- ✗ I computer possono eseguire un ampio spettro di compiti perché possono eseguire diversi programmi
 - Ciascun programma è progettato per indirizzare il computer affinché lavori su un compito specifico
- ✗ Programmazione:
 - È l'atto (e anche l'arte) di progettare, implementare e verificare (testare) i programmi



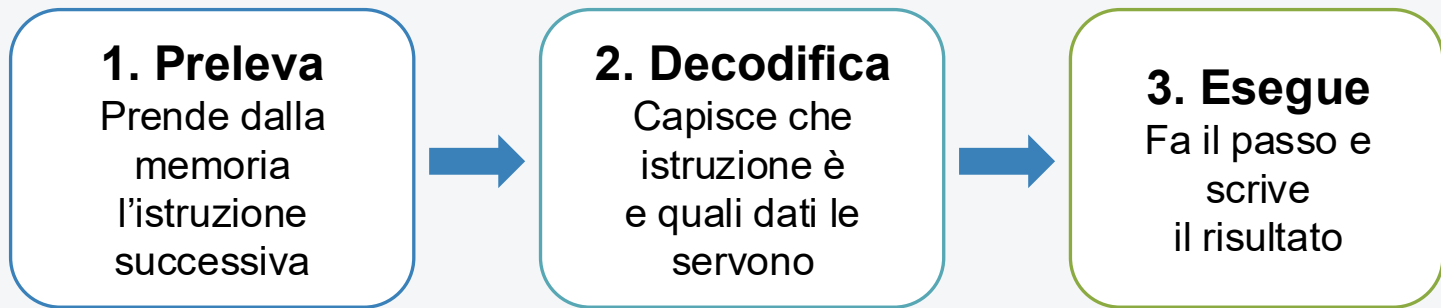
ESEGUIRE UN PROGRAMMA

- ✗ Le istruzioni di un programma ed i relativi dati (come testi, numeri, audio o video) sono memorizzati in formato digitale
- ✗ Per eseguire un programma, questo deve essere portato in memoria, dove la CPU lo possa leggere
- ✗ La CPU esegue il programma un'istruzione per volta
 - Il programma può anche reagire agli input provenienti dall'utente
- ✗ Le istruzioni e gli input dell'utente determinano l'esecuzione del programma
 - La CPU legge i dati (compreso l'input utente), li modifica e li riscrive nuovamente in memoria, sullo schermo, o sulla memoria di massa



LA CPU RIPETE SEMPRE LO STESSO CICLO

Non fa altro, dall'accensione allo spegnimento. Miliardi di volte al secondo, un passo alla volta.



...e poi ricomincia da capo con l'istruzione dopo.

Un processore moderno fa qualche miliardo di giri di questo ciclo al secondo.

Per capire che numero sia: un miliardo di secondi sono trentadue anni.

La macchina non è intelligente. È solo ottusamente, inimmaginabilmente veloce.

ASSEMBLY

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

C000                ORG      RCM+$0000 BEGIN MONITOR
C000 8E 00 70  START  LDS      #STACK

*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013      RESETA EQU    %00010011
0011      CTLREG EQU    %00010001

C003 86 13      INITA  LDA A  #RESETA  RESET ACIA
C005 B7 80 04      STA A  ACIA
C008 86 11      LDA A  #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04      STA A  ACIA

C00D 7E C0 F1      JMP      SIGNON  GO TO START OF MONITOR

*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH  LDA A  ACIA      GET STATUS
C013 47                ASR A          SHIFT RDRF FLAG INTO CARRY
C014 24 FA          BCC  INCH        RECIEVE NOT READY
C016 B6 80 05      LDA A  ACIA+1    GET CHAR
C019 84 7F          AND A  #$7F     MASK PARITY
C01B 7E C0 79      JMP  OUTCH        ECHO & RTS

*****
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input

C01E 8D F0      INHEX  BSR  INCH      GET A CHAR
C020 81 30      CMP A  #'0          ZERO
C022 2B 11      BMI  HEXERR        NOT HEX
C024 81 39      CMP A  #'9          NINE
C026 2F 0A      BLE  HEXRTS        GOOD HEX
C028 81 41      CMP A  #'A          NOT HEX
C02A 2B 09      BMI  HEXERR
C02C 81 46      CMP A  #'F
C02E 2E 05      BGT  HEXERR
C030 80 07      SUB A  #7          FIX A-F
C032 84 0F      HEXRTS AND A  #$0F  CONVERT ASCII TO DIGIT
C034 39                RTS

C035 7E C0 AF  HEXERR JMP  CTRL      RETURN TO CONTROL LOOP
```

Il linguaggio assembly è un linguaggio di programmazione molto simile al linguaggio macchina, pur essendo differente rispetto a quest'ultimo

Assembly ≠ Assembler

Erroneamente viene spesso chiamato "assembler", ma quest'ultimo termine identifica solo il programma "assemblatore" che converte il linguaggio assembly in linguaggio macchina. Le istruzioni del linguaggio assembly corrispondono alle istruzioni che il processore può eseguire.

[Assembly x86](#)

CUCINARE VS PROGRAMMARE

Orange Cake

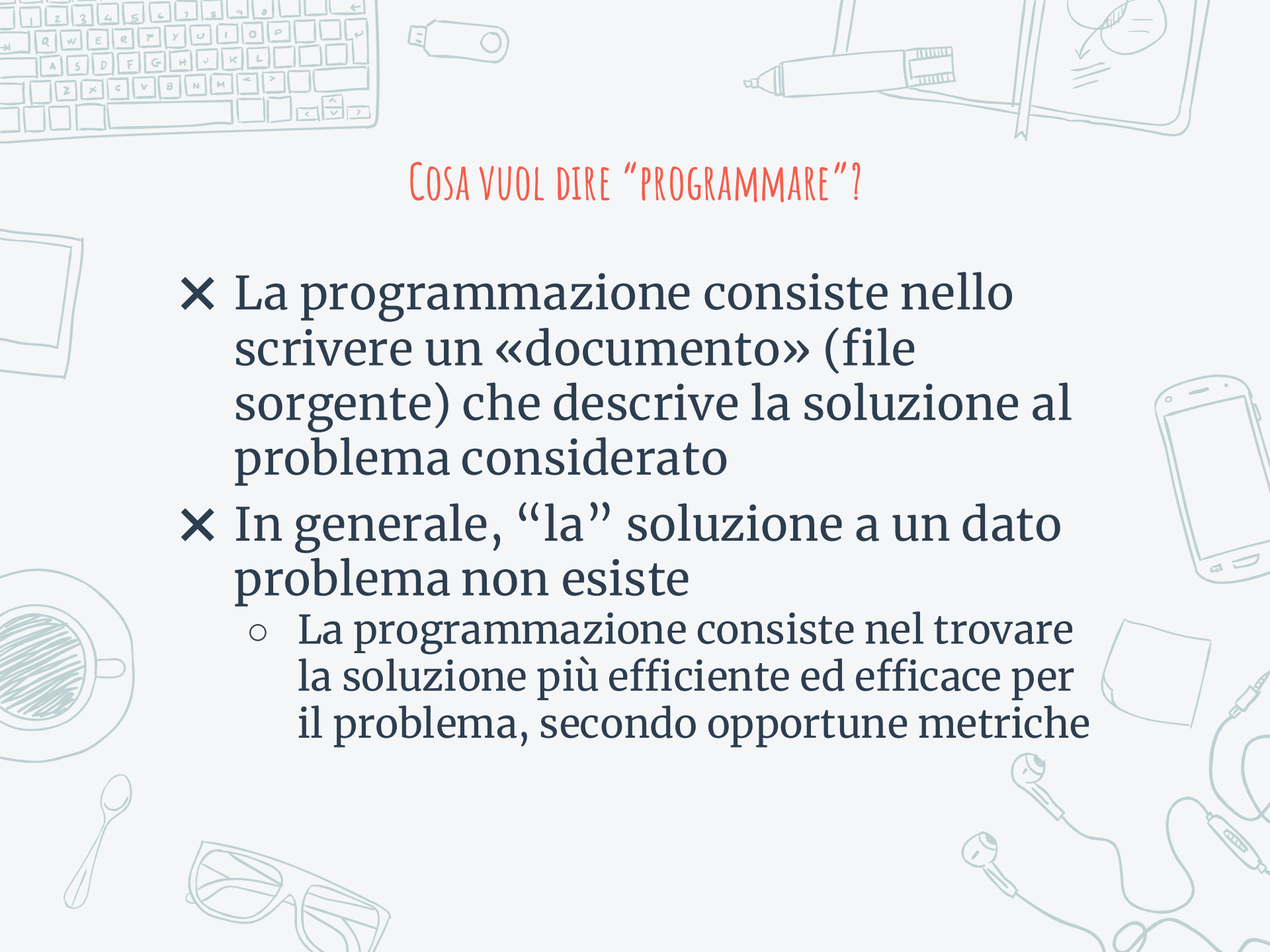
4 ozs. Butter
1 Cup Sugar
2 Cups Flour
1/2 Cup Orange Juice
2 Teaspoons Baking Powder
3 — — — Eggs
Rind of orange

Ingredienti/dati

Cream butter & sugar
add eggs one by one
beating each in well.
Then add grated rind of
orange — flour & juice
separately. Bake in deep
pan. 350°

Istruzioni

```
/* ===== */  
/* MAIN NODE */  
/* ===== */  
  
void CruiseControl::C_CruiseControl * _C_ )  
{  
    bool BrakePressed;  
    bool AcceleratorPressed;  
    bool SpeedOutOfLimits;  
    bool _L19;  
    /* code for node CruiseControl */  
    /* call to node not expanded DetectPedalsPressed */  
    _C_ -> On_DetectPedalsPressed._I0_Brake) = (_C_ -> _C_ -> On_DetectPedalsPressed._I1_Accelerator) = T_0;  
    DetectPedalsPressed(&_C_ -> On_DetectPedalsPressed._I0_BrakePressed = (_C_ -> On_DetectPedalsPressed._I0_AcceleratorPressed =  
    AcceleratorPressed =  
    _C_ -> On_DetectPedalsPressed._I1_AcceleratorPressed =  
    /* call to node not expanded DetectSpeedLimits */  
    _C_ -> On_DetectSpeedLimits._I0_Speed) = (_C_ -> _C_ -> On_DetectSpeedLimits._I1_SpeedOutOfLimits) = T_0;  
    DetectSpeedLimits(&_C_ -> On_DetectSpeedLimits._I1_SpeedOutOfLimits = T_0 -> On_DetectSpeedLimits._I0_SpeedOutOfLimits =  
    /* call to node not expanded CruiseStateMgt */  
    _C_ -> On_CruiseStateMgt._I0_BrakePressed = BrakePressed;
```



COSA VUOL DIRE "PROGRAMMARE"?

- ✗ La programmazione consiste nello scrivere un «documento» (file sorgente) che descrive la soluzione al problema considerato
- ✗ In generale, “la” soluzione a un dato problema non esiste
 - La programmazione consiste nel trovare la soluzione più efficiente ed efficace per il problema, secondo opportune metriche



COSA VUOL DIRE 'PROGRAMMARE'?

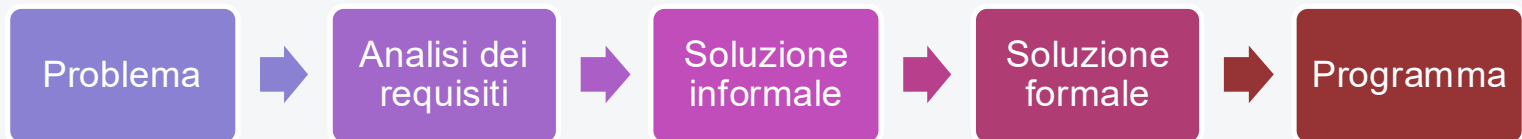
✗ Programmare è un'attività 'creativa'

- Ogni problema è diverso da ogni altro problema
- Non ci sono soluzioni sistematiche/analitiche o soluzioni universali

✗ Programmare è un'operazione complessa

- Un approccio diretto è impraticabile o comunque molto difficile per compiti non banali

SVILUPPO DI UN PROGRAMMA



Analisi dei requisiti — che cosa deve fare, davvero. ▶ Soluzione informale — a parole, come lo risolverei io. ▶ Soluzione formale — l'algoritmo, non ambiguo. ▶ Programma — la traduzione.

Scrivere il codice è l'ultimo passo, ed è il più corto.

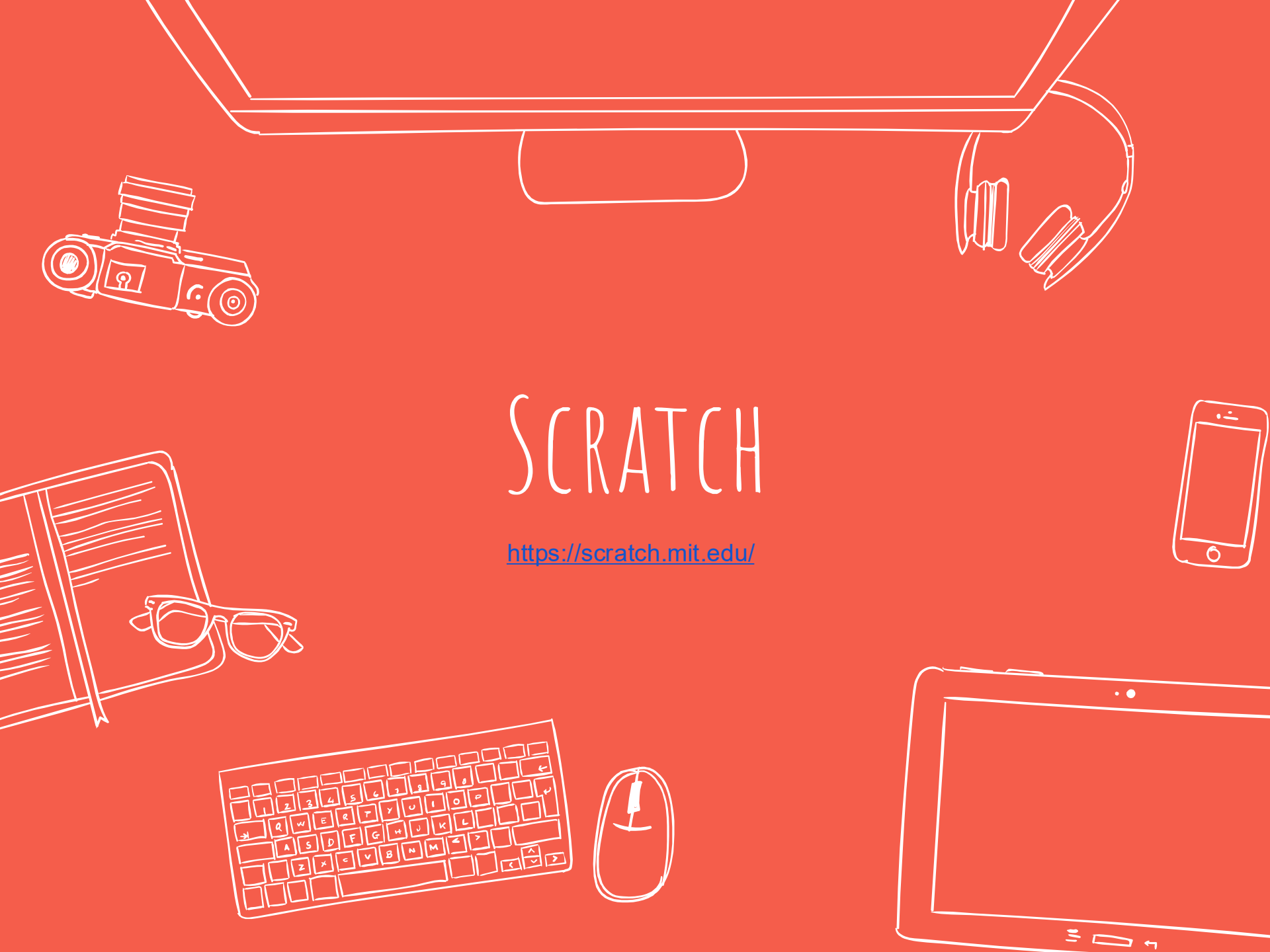


CODE.ORG

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

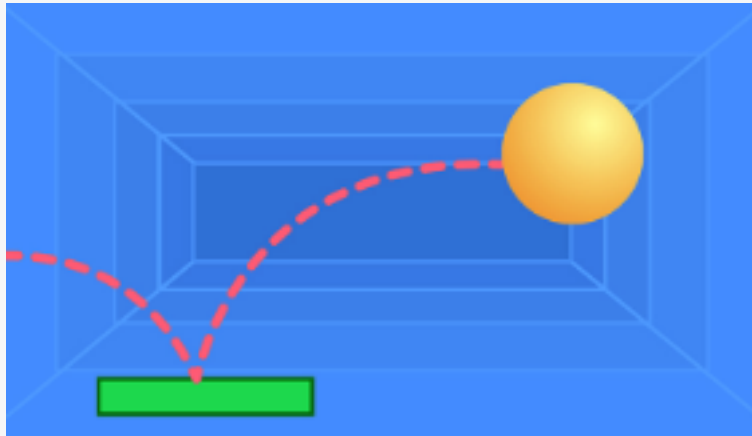
SCRATCH

<https://scratch.mit.edu/>



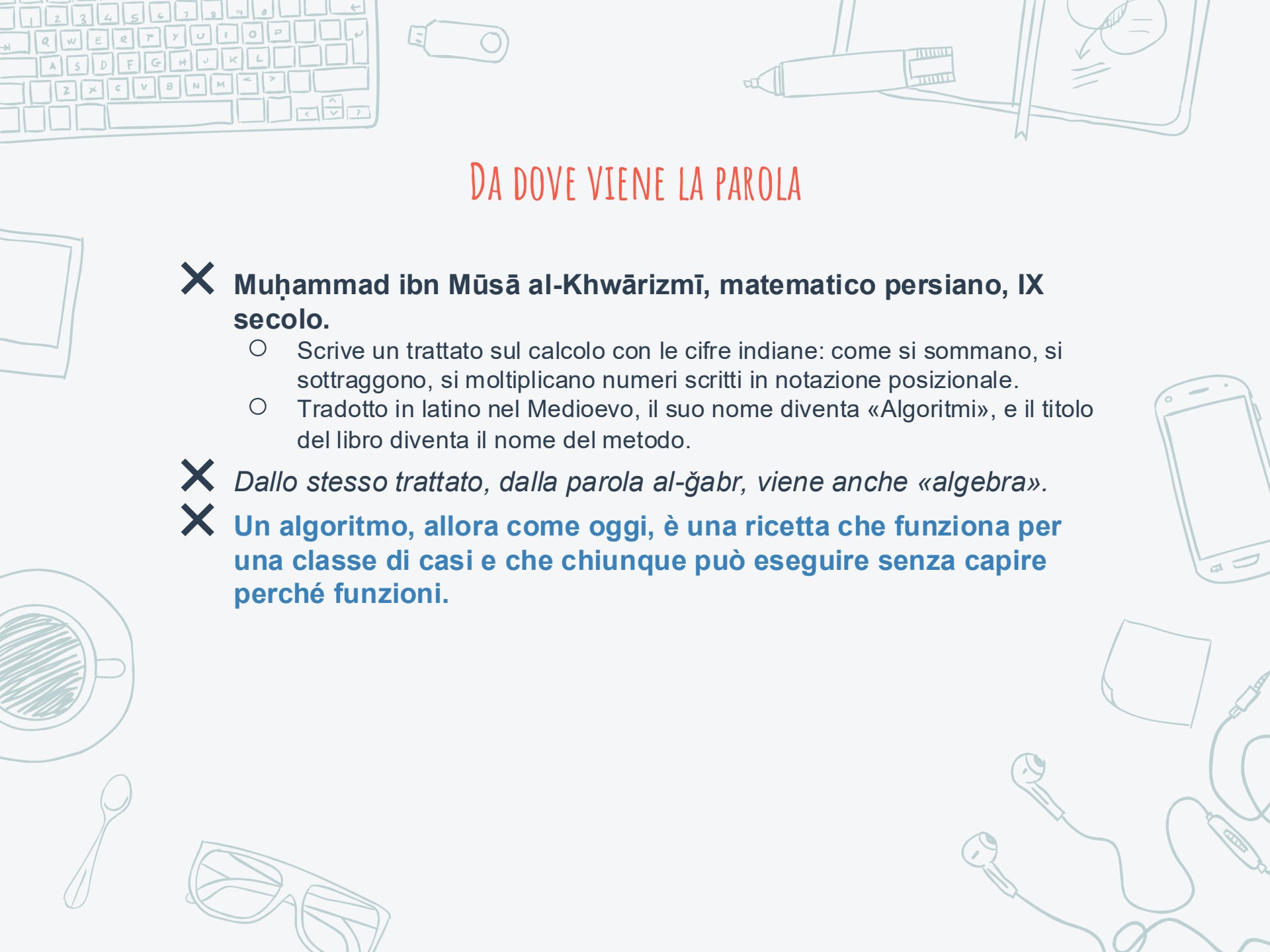
PONG

<https://scratch.mit.edu/projects/editor/?tutorial=pong>



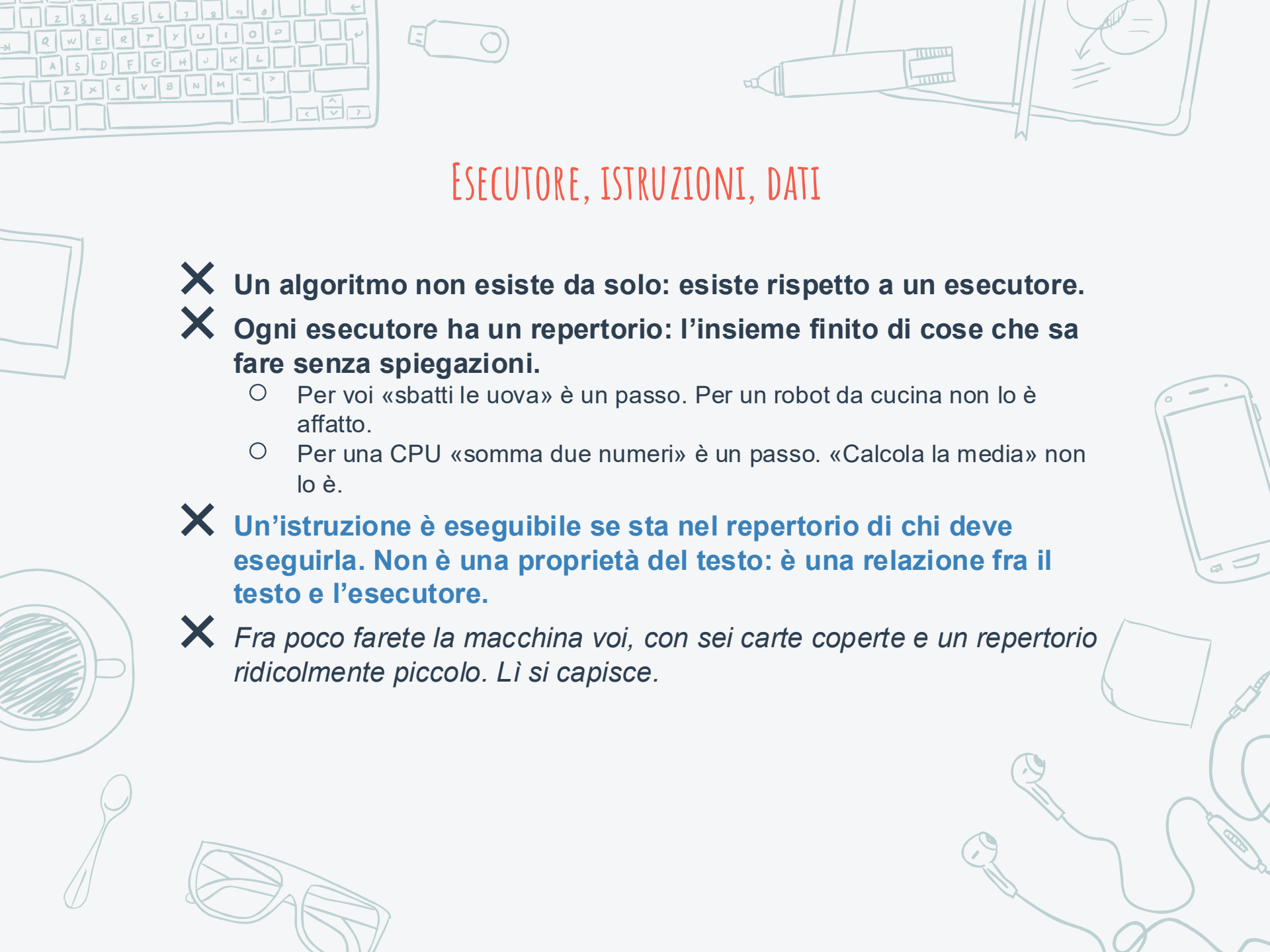
L'ALGORITMO





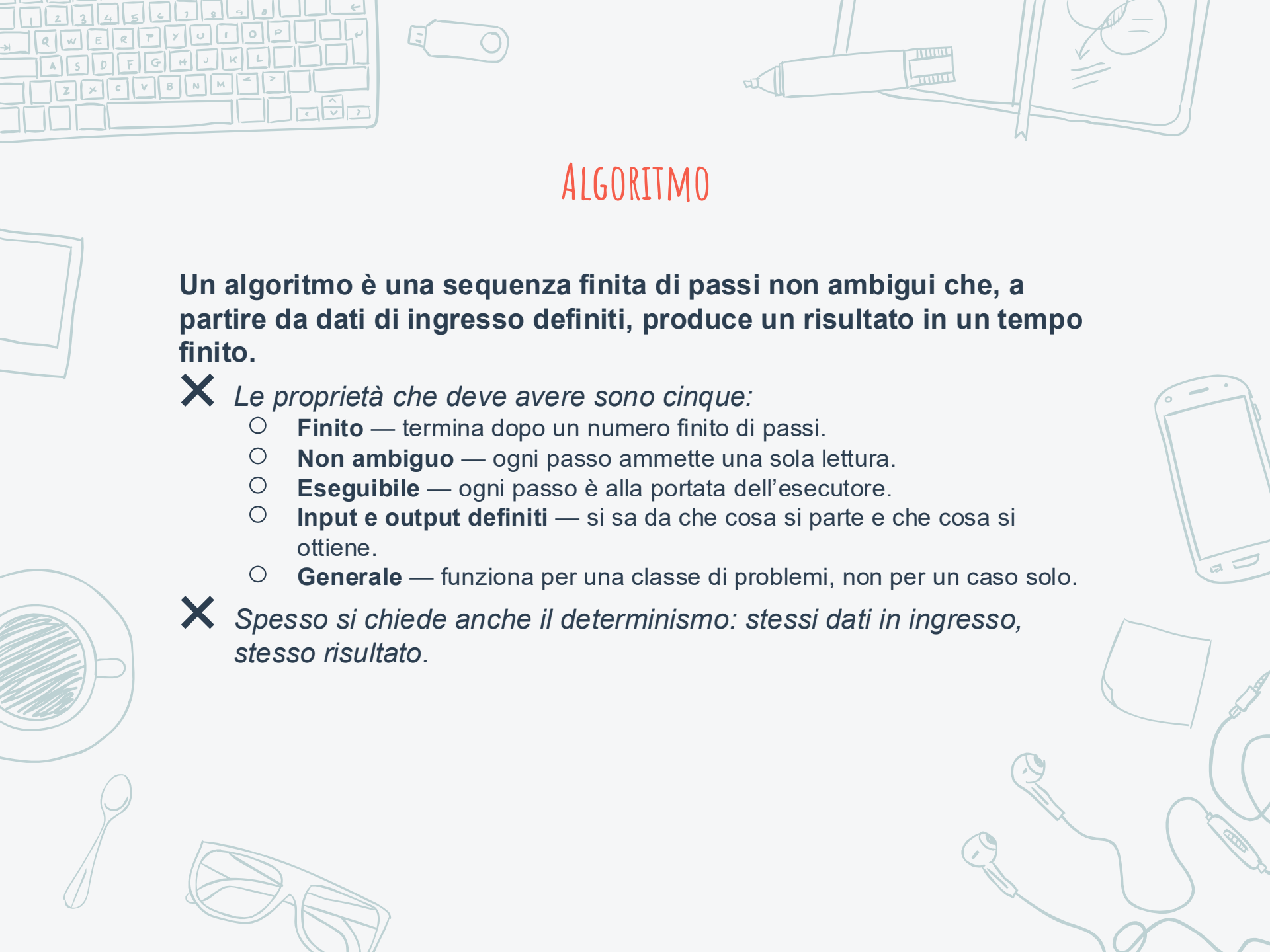
DA DOVE VIENE LA PAROLA

- ✗ **Muḥammad ibn Mūsā al-Khwārizmī, matematico persiano, IX secolo.**
 - Scrive un trattato sul calcolo con le cifre indiane: come si sommano, si sottraggono, si moltiplicano numeri scritti in notazione posizionale.
 - Tradotto in latino nel Medioevo, il suo nome diventa «Algoritmi», e il titolo del libro diventa il nome del metodo.
- ✗ *Dallo stesso trattato, dalla parola al-ğabr, viene anche «algebra».*
- ✗ **Un algoritmo, allora come oggi, è una ricetta che funziona per una classe di casi e che chiunque può eseguire senza capire perché funzioni.**



ESECUTORE, ISTRUZIONI, DATI

- ✗ **Un algoritmo non esiste da solo: esiste rispetto a un esecutore.**
- ✗ **Ogni esecutore ha un repertorio: l'insieme finito di cose che sa fare senza spiegazioni.**
 - Per voi «sbatti le uova» è un passo. Per un robot da cucina non lo è affatto.
 - Per una CPU «somma due numeri» è un passo. «Calcola la media» non lo è.
- ✗ **Un'istruzione è eseguibile se sta nel repertorio di chi deve eseguirla. Non è una proprietà del testo: è una relazione fra il testo e l'esecutore.**
- ✗ *Fra poco farete la macchina voi, con sei carte coperte e un repertorio ridicolmente piccolo. Lì si capisce.*



ALGORITMO

Un algoritmo è una sequenza finita di passi non ambigui che, a partire da dati di ingresso definiti, produce un risultato in un tempo finito.

✗ *Le proprietà che deve avere sono cinque:*

- **Finito** — termina dopo un numero finito di passi.
- **Non ambiguo** — ogni passo ammette una sola lettura.
- **Eseguibile** — ogni passo è alla portata dell'esecutore.
- **Input e output definiti** — si sa da che cosa si parte e che cosa si ottiene.
- **Generale** — funziona per una classe di problemi, non per un caso solo.

✗ *Spesso si chiede anche il determinismo: stessi dati in ingresso, stesso risultato.*

CINQUE PROPRIETÀ DA CONTROLLARE SEMPRE

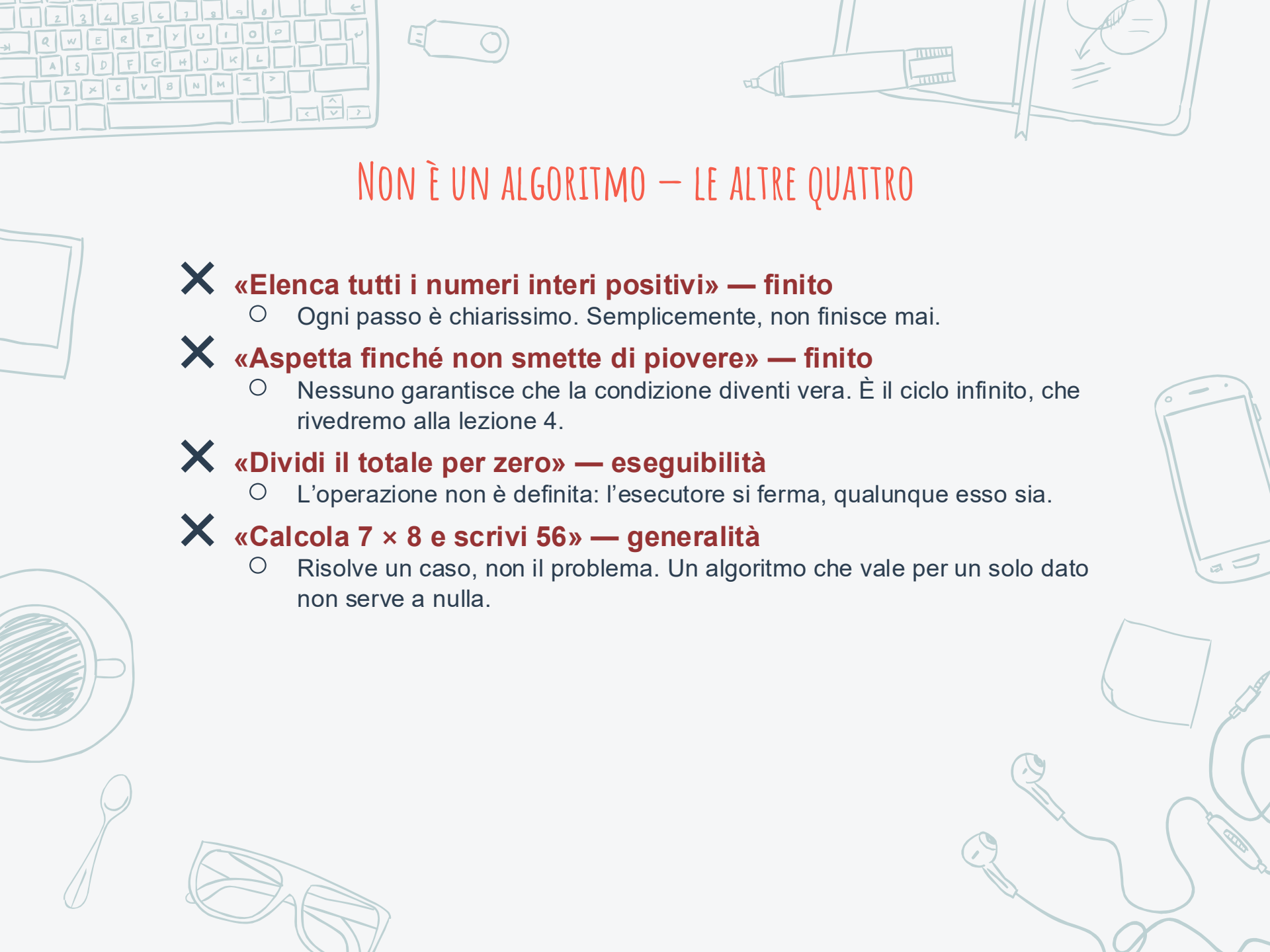
Proprietà	Che cosa richiede	Come si viola
Finito	Termina dopo passi finiti	«Aspetta finché non smette di piovere»
Non ambiguo	Un passo, una sola lettura	«Aggiungi sale quanto basta»
Eseguibile	Ogni passo è nel repertorio	«Indovina la password»
Input / output	Dati iniziali e risultato definiti	«Calcola la media» — di che cosa?
Generale	Vale per una classe di casi	«Il risultato è 56» — e con altri numeri?

Quando una procedura non funziona, la domanda non è «dov'è l'errore?». È «quale di queste cinque ho violato?».



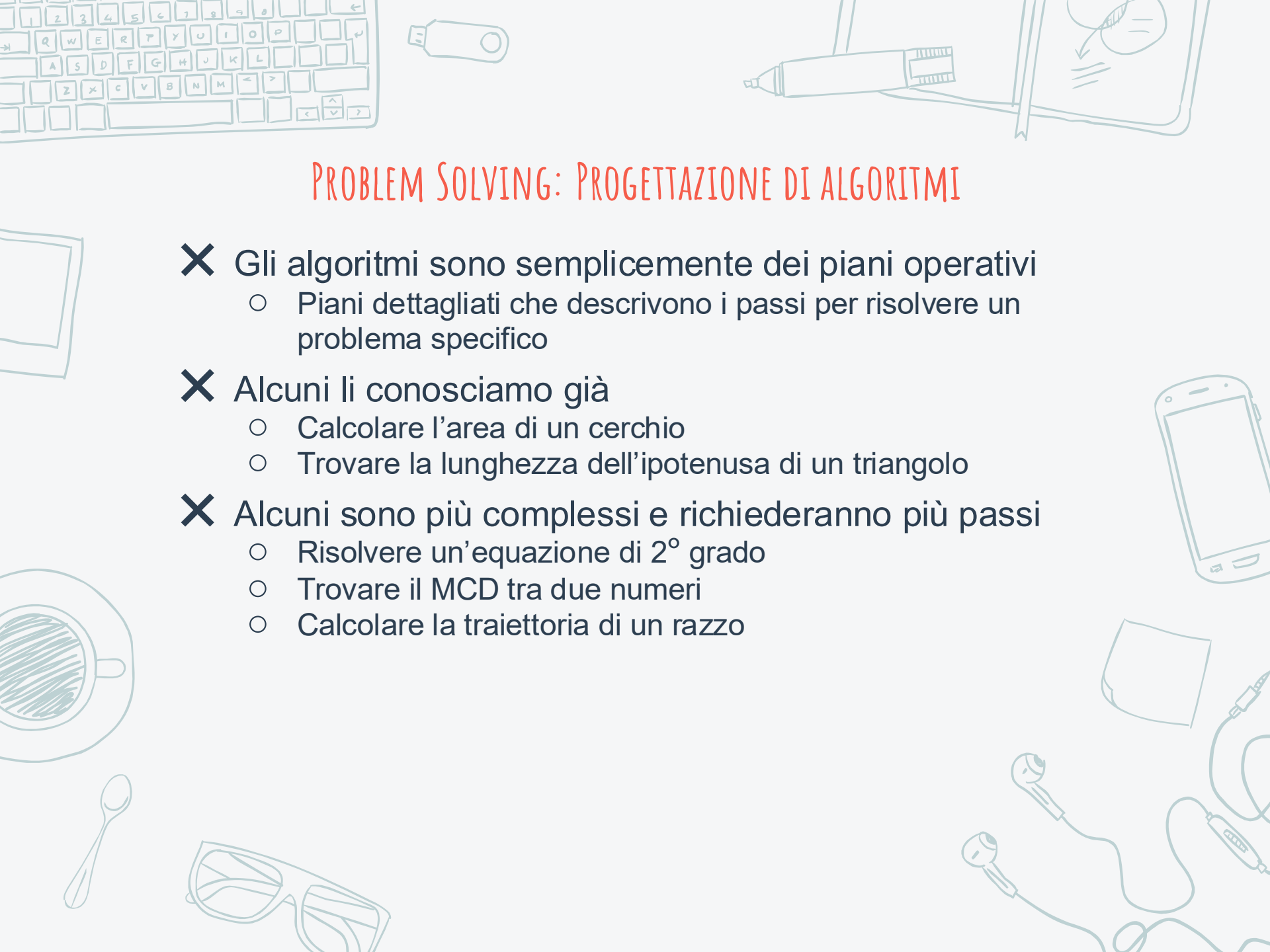
NON È UN ALGORITMO — AMBIGUITÀ

- ✗ **«Aggiungi sale quanto basta»**
 - Quanto basta a chi? Due esecutori, due piatti diversi. Il passo non è sbagliato: è indeterminato.
- ✗ **«Cuoci il risotto come fa la nonna»**
 - Presuppone una conoscenza che l'esecutore non ha. Un algoritmo non può rimandare a un sapere non scritto.
- ✗ **«Fai la media dei voti»**
 - Descrive il risultato, non il procedimento. È un obiettivo travestito da istruzione — l'errore più comune, e il più difficile da vedere.



NON È UN ALGORITMO — LE ALTRE QUATTRO

- ✗ **«Elenca tutti i numeri interi positivi» — finito**
 - Ogni passo è chiarissimo. Semplicemente, non finisce mai.
- ✗ **«Aspetta finché non smette di piovere» — finito**
 - Nessuno garantisce che la condizione diventi vera. È il ciclo infinito, che rivedremo alla lezione 4.
- ✗ **«Dividi il totale per zero» — eseguibilità**
 - L'operazione non è definita: l'esecutore si ferma, qualunque esso sia.
- ✗ **«Calcola 7×8 e scrivi 56» — generalità**
 - Risolve un caso, non il problema. Un algoritmo che vale per un solo dato non serve a nulla.



PROBLEM SOLVING: PROGETTAZIONE DI ALGORITMI

- ✗ Gli algoritmi sono semplicemente dei piani operativi
 - Piani dettagliati che descrivono i passi per risolvere un problema specifico
- ✗ Alcuni li conosciamo già
 - Calcolare l'area di un cerchio
 - Trovare la lunghezza dell'ipotenusa di un triangolo
- ✗ Alcuni sono più complessi e richiederanno più passi
 - Risolvere un'equazione di 2° grado
 - Trovare il MCD tra due numeri
 - Calcolare la traiettoria di un razzo

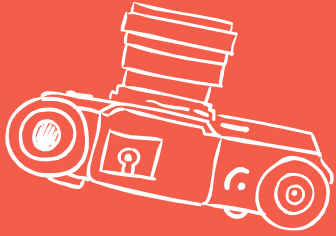
ALGORITMI NELLA VITA QUOTIDIANA

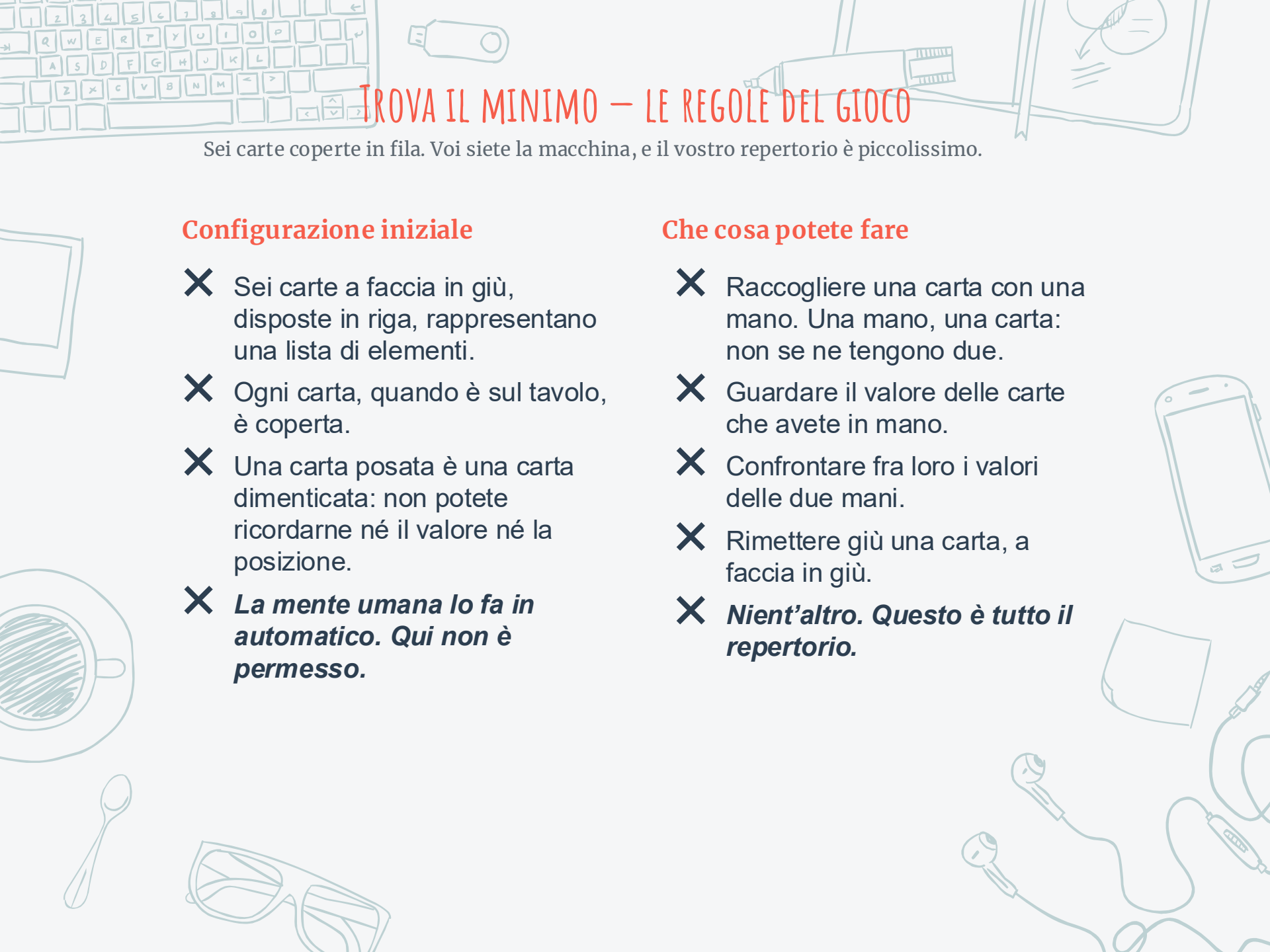
1. Metti l'acqua
2. Accendi il fuoco
3. Aspetta
4. Se l'acqua non bolle
torna a 3
5. Butta la pasta
6. Aspetta un po'
7. Assaggia
8. Se è cruda torna a 6
9. Scola la pasta



CHI SI È ACCORTO CHE
MANCA IL SALE?

TROVA IL MINIMO





TROVA IL MINIMO — LE REGOLE DEL GIOCO

Sei carte coperte in fila. Voi siete la macchina, e il vostro repertorio è piccolissimo.

Configurazione iniziale

- ✗ Sei carte a faccia in giù, disposte in riga, rappresentano una lista di elementi.
- ✗ Ogni carta, quando è sul tavolo, è coperta.
- ✗ Una carta posata è una carta dimenticata: non potete ricordarne né il valore né la posizione.
- ✗ ***La mente umana lo fa in automatico. Qui non è permesso.***

Che cosa potete fare

- ✗ Raccogliere una carta con una mano. Una mano, una carta: non se ne tengono due.
- ✗ Guardare il valore delle carte che avete in mano.
- ✗ Confrontare fra loro i valori delle due mani.
- ✗ Rimettere giù una carta, a faccia in giù.
- ✗ ***Nient'altro. Questo è tutto il repertorio.***

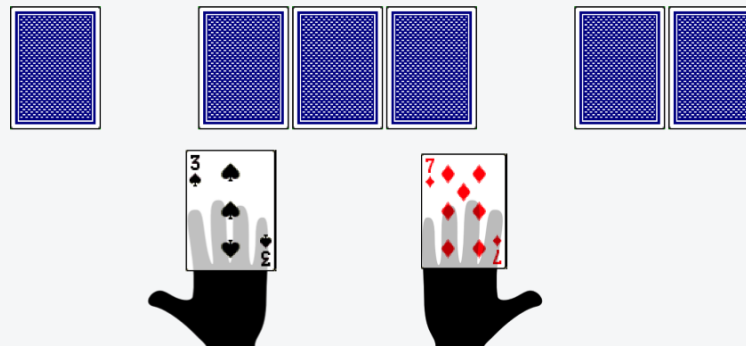
TROVA IL MINIMO – L'OBIETTIVO

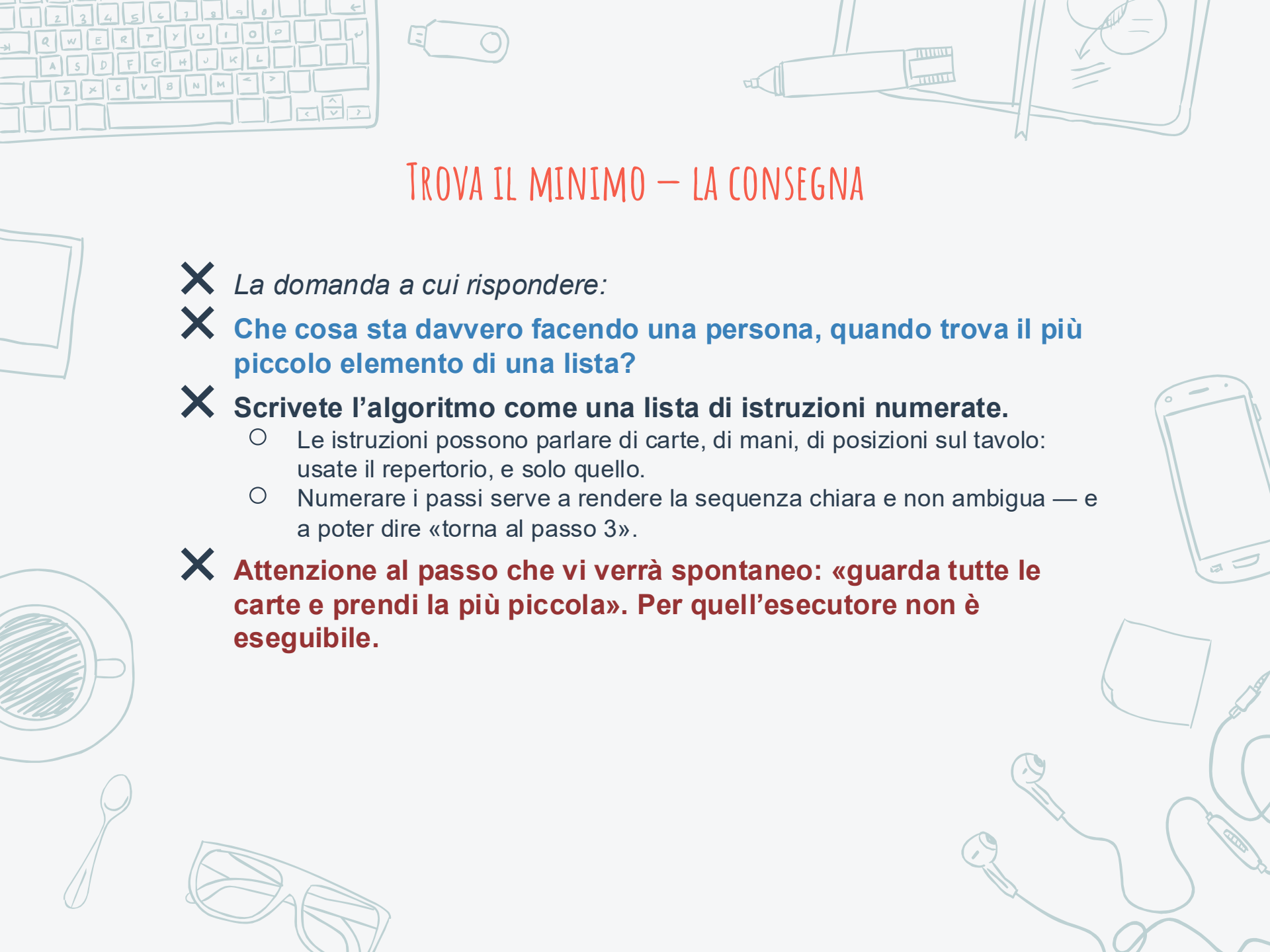
✗ L'algoritmo deve terminare in modo chiaro.

- L'ultima istruzione è «Carta trovata!», e in quel momento avete in mano la carta di valore minimo.

✗ Deve funzionare con qualunque numero di carte.

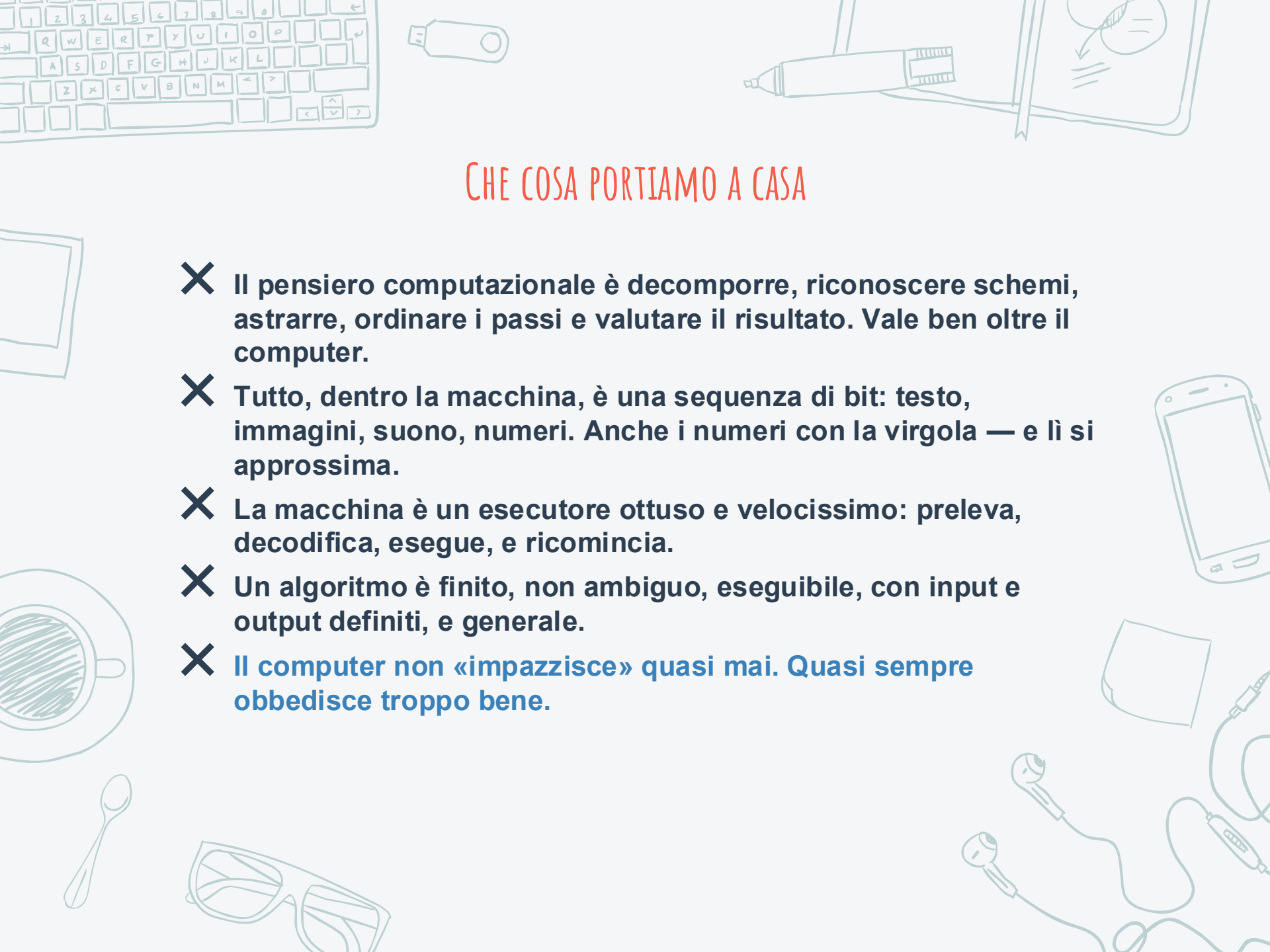
- Con 1, con 6 o con 1 000. Se avete scritto «guarda la terza», avete scritto la soluzione di un caso, non un algoritmo.





TROVA IL MINIMO — LA CONSEGNA

- ✗ *La domanda a cui rispondere:*
- ✗ **Che cosa sta davvero facendo una persona, quando trova il più piccolo elemento di una lista?**
- ✗ **Scrivete l'algoritmo come una lista di istruzioni numerate.**
 - Le istruzioni possono parlare di carte, di mani, di posizioni sul tavolo: usate il repertorio, e solo quello.
 - Numerare i passi serve a rendere la sequenza chiara e non ambigua — e a poter dire «torna al passo 3».
- ✗ **Attenzione al passo che vi verrà spontaneo: «guarda tutte le carte e prendi la più piccola». Per quell'esecutore non è eseguibile.**



CHE COSA PORTIAMO A CASA

- ✗ Il pensiero computazionale è decomporre, riconoscere schemi, astrarre, ordinare i passi e valutare il risultato. Vale ben oltre il computer.
- ✗ Tutto, dentro la macchina, è una sequenza di bit: testo, immagini, suono, numeri. Anche i numeri con la virgola — e lì si approssima.
- ✗ La macchina è un esecutore ottuso e velocissimo: preleva, decodifica, esegue, e ricomincia.
- ✗ Un algoritmo è finito, non ambiguo, eseguibile, con input e output definiti, e generale.
- ✗ Il computer non «impazzisce» quasi mai. Quasi sempre obbedisce troppo bene.