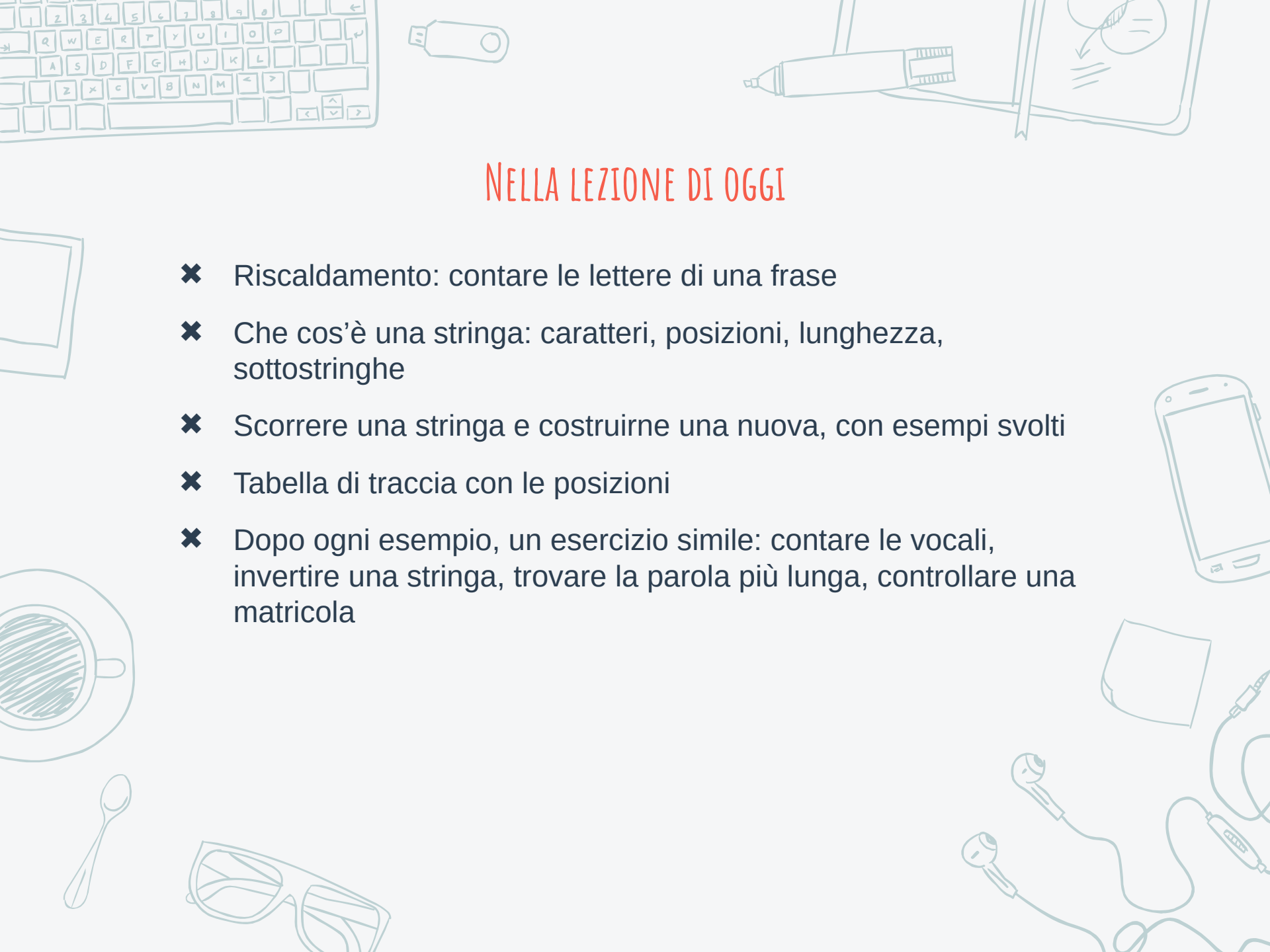




LABORATORIO DI PENSIERO COMPUTAZIONALE

Lezione 6 — Le stringhe

Simone Zennaro – simone.zennaro@unipd.it



NELLA LEZIONE DI OGGI

- ✘ Riscaldamento: contare le lettere di una frase
- ✘ Che cos'è una stringa: caratteri, posizioni, lunghezza, sottostringhe
- ✘ Scorrere una stringa e costruirne una nuova, con esempi svolti
- ✘ Tabella di traccia con le posizioni
- ✘ Dopo ogni esempio, un esercizio simile: contare le vocali, invertire una stringa, trovare la parola più lunga, controllare una matricola

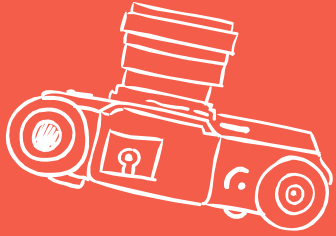
DATI CHE NON SONO NUMERI

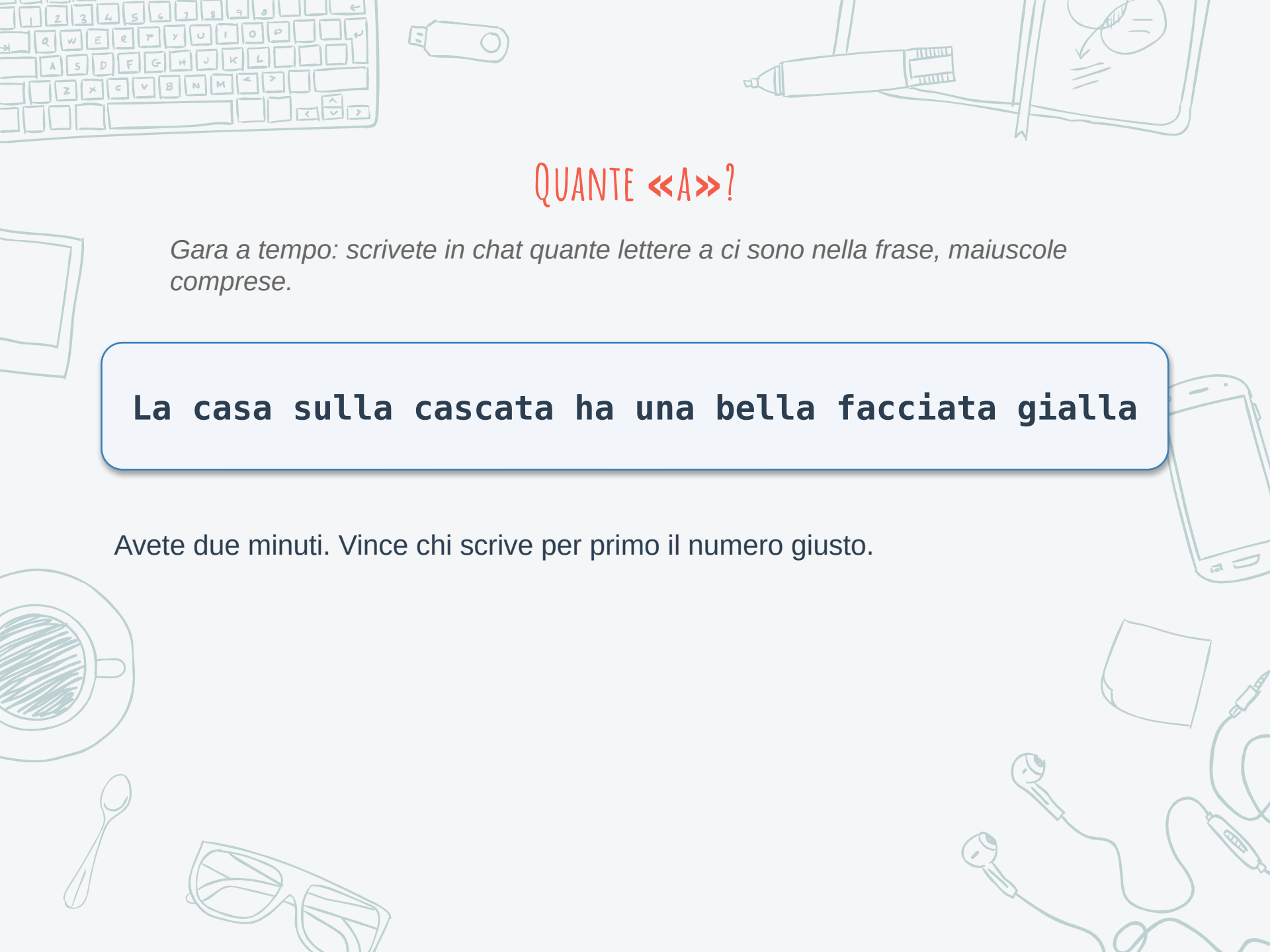
Finora gli algoritmi hanno lavorato su numeri. Molti dati, però, sono testi.

Dato	Esempio	Una domanda che ci interessa
Nome	Ada Lovelace	Quante vocali contiene?
Parola	Roma	Come si legge al contrario?
Frase	il sole splende	Qual è la parola più lunga?
Matricola	AB123456	È scritta nel formato giusto?
Password	Statistica26!	Rispetta le regole?

Per rispondere, un algoritmo deve esaminare il testo un carattere alla volta. Le prime quattro domande sono gli esercizi di oggi; la password è la sfida facoltativa.

RISCALDAMENTO



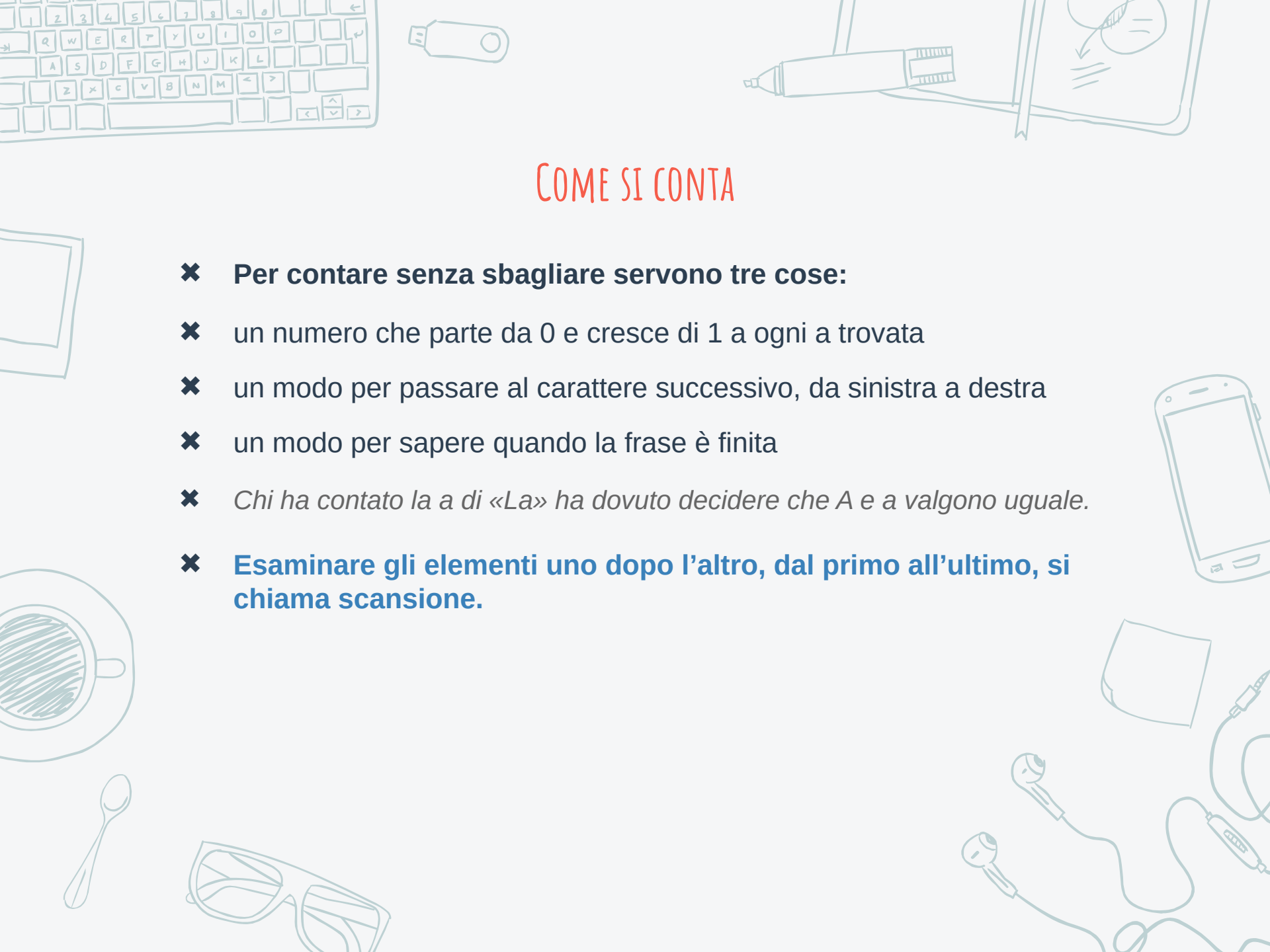


QUANTE «A»?

Gara a tempo: scrivete in chat quante lettere a ci sono nella frase, maiuscole comprese.

La casa sulla cascata ha una bella facciata gialla

Avete due minuti. Vince chi scrive per primo il numero giusto.



COME SI CONTA

- ✗ **Per contare senza sbagliare servono tre cose:**
- ✗ un numero che parte da 0 e cresce di 1 a ogni trovata
- ✗ un modo per passare al carattere successivo, da sinistra a destra
- ✗ un modo per sapere quando la frase è finita
- ✗ *Chi ha contato la a di «La» ha dovuto decidere che A e a valgono uguale.*
- ✗ **Esaminare gli elementi uno dopo l'altro, dal primo all'ultimo, si chiama scansione.**

LE STRINGHE



CHE COS'È UNA STRINGA

Una stringa è un dato che contiene un testo. Si scrive fra virgolette.

Stringa	Che cosa rappresenta
"Ada"	un nome
"AB123456"	una matricola
"il sole splende"	una frase
"26"	un testo di due caratteri
""	la stringa vuota

Si usa come gli altri dati

```
nome = "Ada"  
saluto = "Ciao"  
scrivi saluto  
scrivi nome
```

Sullo schermo compare:

Ciao
Ada

Le virgolette separano il testo dai nomi delle variabili: nome è una variabile, "nome" è un testo di quattro lettere.

UNA SEQUENZA DI CARATTERI

Una stringa è formata da caratteri messi in fila: "Ciao! 26".

C	i	a	o	!	␣	2	6
0	1	2	3	4	5	6	7

Ogni carattere occupa una posizione. Le posizioni partono da 0.

Anche lo spazio, il punto esclamativo e le cifre sono caratteri. "26" è una stringa di due caratteri, non il numero 26.

"" è la stringa vuota: non contiene nessun carattere e la sua lunghezza è 0.

Per il computer una stringa è una sequenza ordinata, come una fila di caselle numerate.

LUNGHEZZA E POSIZIONI

La notazione che useremo nello pseudocodice.

Scrittura	Significato	Con s = "Roma"
lunghezza(s)	numero di caratteri	4
s[i]	carattere in posizione i	s[0] = "R", s[3] = "a"
s[lunghezza(s) - 1]	ultimo carattere	"a"
s[4]	posizione che non esiste	errore

R	o	m	a
0	1	2	3

Una stringa lunga n ha le posizioni da 0 a n - 1.

SOTTOSTRINGHE

Una sottostringa è un pezzo di una stringa formato da caratteri consecutivi.

`s[inizio:fine]`

inizio: la prima posizione presa (inclusa)

fine: la prima posizione non presa (esclusa)

Con `s = "informatica"`

i	n	f	o	r	m	a	t	i	c	a
0	1	2	3	4	5	6	7	8	9	10

`s[0:5] = "infor"`

Prende le posizioni 0, 1, 2, 3, 4. Il carattere in posizione 5 («m») resta fuori.

La sottostringa `s[inizio:fine]` ha `fine - inizio` caratteri: `s[0:5]` ne ha 5. In Python si scrive nello stesso modo.

SOTTOSTRINGHE: ESEMPI

Sempre con $s = \text{"informatica"}$, lunga 11.

Scrittura	Risultato	Perché
<code>s[0:5]</code>	<code>"infor"</code>	posizioni da 0 a 4
<code>s[5:11]</code>	<code>"matica"</code>	da 5 fino all'ultima: 11 è la lunghezza
<code>s[2:3]</code>	<code>"f"</code>	un solo carattere, lo stesso di <code>s[2]</code>
<code>s[4:4]</code>	<code>""</code>	inizio uguale a fine: nessun carattere
<code>s[:4]</code>	<code>"info"</code>	senza inizio si parte da 0
<code>s[5:]</code>	<code>"matica"</code>	senza fine si arriva all'ultimo carattere

Nella matricola `"AB123456"`: `s[0:2] = "AB"` sono le lettere, `s[2:8] = "123456"` sono le cifre.

Se inizio o fine superano la lunghezza, lo pseudocodice non lo permette. Usate solo posizioni da 0 a `lunghezza(s)`.

CONFRONTARE CARATTERI

Nella lezione 1: ogni carattere è memorizzato come un numero (codice ASCII).

Carattere	" "	"0"	"9"	"A"	"Z"	"a"	"z"
Codice	32	48	57	65	90	97	122

I caratteri si confrontano come i loro codici:

"A" < "B" è vero, "Z" < "a" è vero, "a" = "A" è falso

Per sapere se c è una cifra basta una condizione composta (lezione 3):

$c \geq "0"$ e $c \leq "9"$

Per sapere se c è una vocale si può scrivere:

c è in "aeiouAEIOU"

Maiuscole e minuscole sono caratteri diversi: "Ciao" = "ciao" è falso.

SCORRERE UNA STRINGA

Due modi di scrivere la scansione. Fanno la stessa cosa.

Con la posizione

```
i = 0
fintanto che i < lunghezza(s):
    c = s[i]
    ... esamina c ...
    i = i + 1
```

Senza la posizione

per ogni carattere c di s:
... esamina c ...

La condizione è $i < \text{lunghezza}(s)$, non $\leq$: l'ultima posizione è $\text{lunghezza}(s) - 1$.

Se serve sapere dove si trova il carattere, o guardare quelli vicini, si usa la prima forma. Se basta guardare un carattere alla volta, la seconda è più corta.

ESEMPIO — CONTARE GLI SPAZI

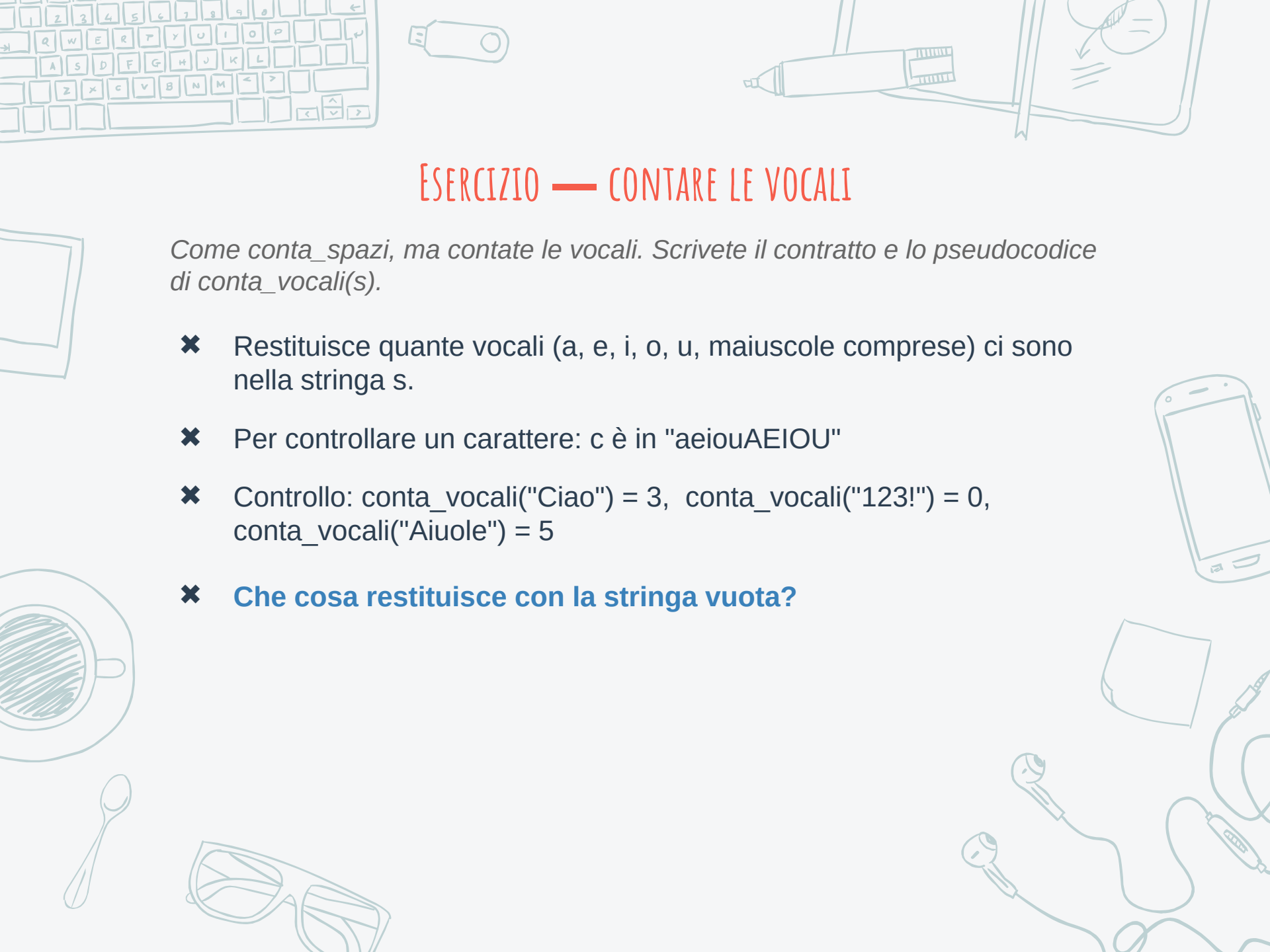
Lo svolgiamo insieme: quanti spazi ci sono nella stringa s?

```
funzione conta_spazi(s):  
    n = 0  
    i = 0  
    fintanto che i < lunghezza(s):  
        se s[i] = " " allora  
            n = n + 1  
        i = i + 1  
    restituisci n
```

Con s = "a b c"

i	i < 5 ?	s[i]	Spazio?	n
0	Vero	a	no	0
1	Vero	␣	sì	1
2	Vero	b	no	1
3	Vero	␣	sì	2
4	Vero	c	no	2
5	Falso	—	—	restituisci 2

Con «per ogni carattere c di s» la stessa funzione è più corta: non serve i, e al posto di s[i] si scrive c.



ESERCIZIO — CONTARE LE VOCALI

Come conta_spazi, ma contate le vocali. Scrivete il contratto e lo pseudocodice di conta_vocali(s).

- ✘ Restituisce quante vocali (a, e, i, o, u, maiuscole comprese) ci sono nella stringa s.
- ✘ Per controllare un carattere: c è in "aeiouAEIOU"
- ✘ Controllo: `conta_vocali("Ciao") = 3`, `conta_vocali("123!") = 0`, `conta_vocali("Aiuole") = 5`
- ✘ **Che cosa restituisce con la stringa vuota?**

SOLUZIONE — VOCALI

Contratto: conta_vocali(s) riceve una stringa e restituisce il numero di vocali che contiene.

Pseudocodice

```
funzione conta_vocali(s):  
    n = 0  
    per ogni carattere c di s:  
        se c è in "aeiouAEIOU" allora  
            n = n + 1  
    restituisci n
```

Python

```
def conta_vocali(s):  
    n = 0  
    for c in s:  
        if c in "aeiouAEIOU":  
            n = n + 1  
    return n
```

s	"Ciao"	"123!"	"Aiuole"	""
conta_vocali(s)	3	0	5	0

Il numero di controlli è uguale alla lunghezza della stringa.

COSTRUIRE UNA STRINGA

Il simbolo + fra due stringhe le unisce in una sola (concatenazione).

Espressione	Risultato
"Ciao" + " " + "Ada"	"Ciao Ada"
"ab" + "c"	"abc"
"c" + "ab"	"cab"
"2" + "6"	"26" (non 8)

Accumulatore

```
r = ""  
per ogni carattere c di s:  
    r = r + c
```

Parte dalla stringa vuota e aggiunge un carattere alla volta. Alla fine r è uguale a s.

È lo schema della somma della lezione 4: al posto di 0 si parte da "", al posto di + fra numeri si usa + fra stringhe. L'ordine conta.

ESEMPIO — TOGLIERE GLI SPAZI

Lo svolgiamo insieme: costruire una nuova stringa uguale a s ma senza spazi.

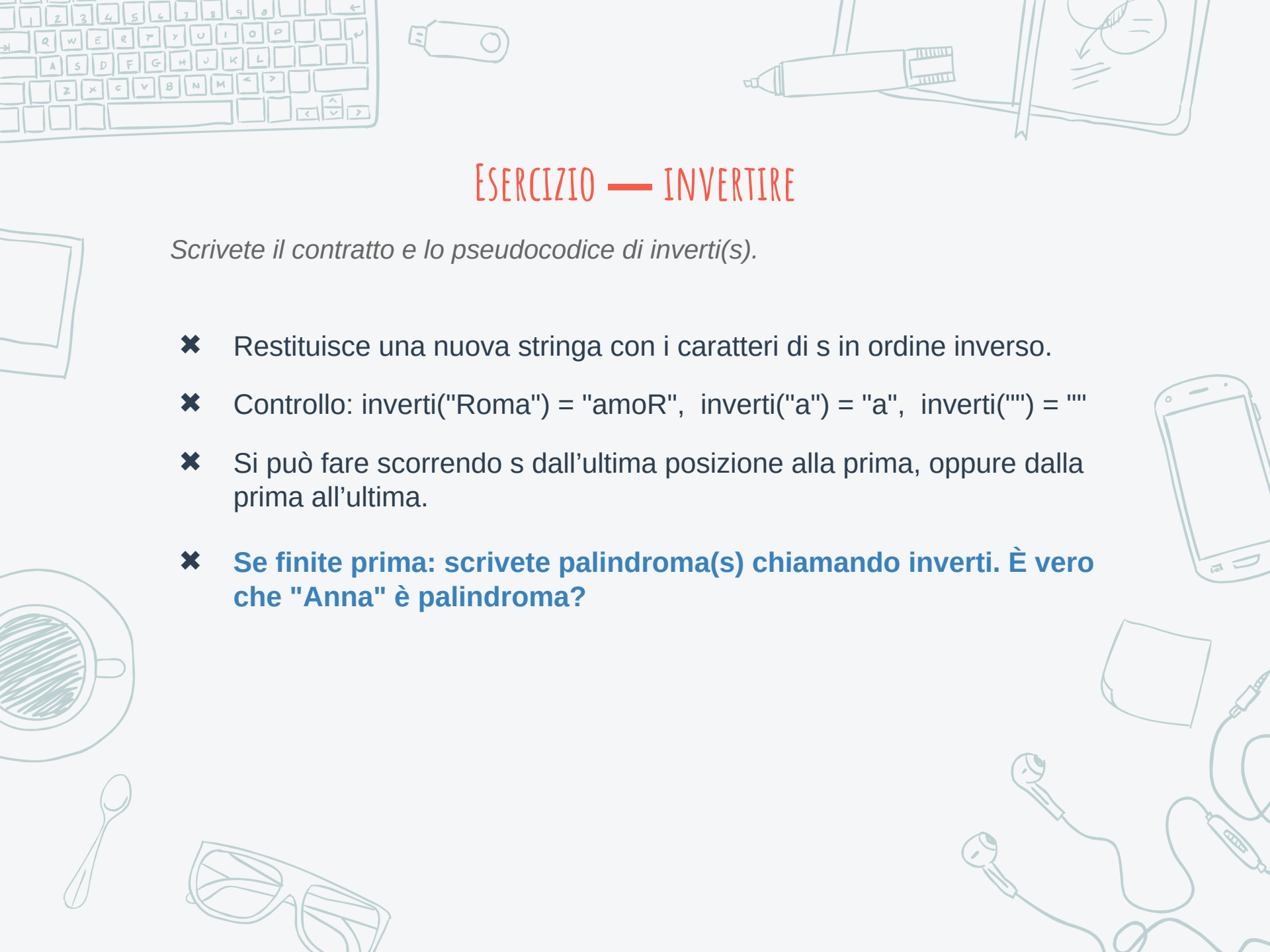
```
funzione toglispazi(s):  
    r = ""  
    per ogni carattere c di s:  
        se c ≠ " " allora  
            r = r + c  
    restituisci r
```

r parte vuota e cresce di un carattere solo quando c non è uno spazio.

Con s = "a b c"

c	a		b		c
r	"a"	"a"	"ab"	"ab"	"abc"

toglispazi("a b c") restituisce "abc". La stringa s non cambia: il risultato è una stringa nuova.



ESERCIZIO — INVERTIRE

Scrivete il contratto e lo pseudocodice di `inverti(s)`.

- ✘ Restituisce una nuova stringa con i caratteri di `s` in ordine inverso.
- ✘ Controllo: `inverti("Roma") = "amoR"`, `inverti("a") = "a"`, `inverti("") = ""`
- ✘ Si può fare scorrendo `s` dall'ultima posizione alla prima, oppure dalla prima all'ultima.
- ✘ **Se finite prima: scrivete `palindroma(s)` chiamando `inverti`. È vero che "Anna" è palindroma?**

SOLUZIONE — INVERTIRE

Contratto: `inverti(s)` riceve una stringa e restituisce i suoi caratteri in ordine inverso.

Pseudocodice

```
funzione inverti(s):  
    r = ""  
    per ogni carattere c di s:  
        r = c + r  
    restituisci r
```

Python

```
def inverti(s):  
    r = ""  
    for c in s:  
        r = c + r  
    return r
```

c	R	o	m	a
r	"R"	"oR"	"moR"	"amoR"

Ogni carattere nuovo va davanti a quelli già presi. Con `r = r + c` si ricopia `s` così com'è.

ESEMPIO — SEPARARE LE PAROLE

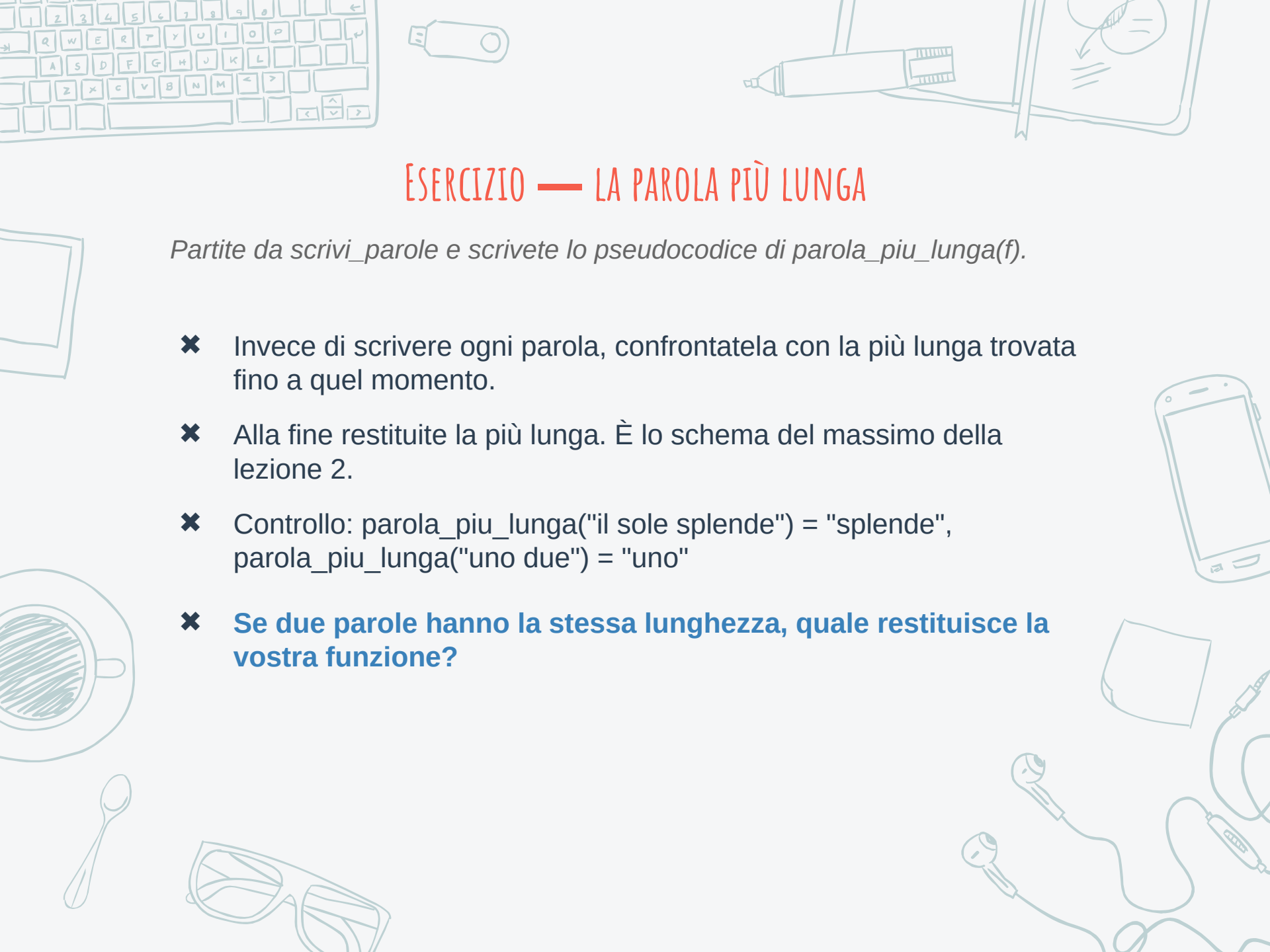
Lo svolgiamo insieme: scrivere una parola per riga. Le parole sono separate da uno spazio.

```
funzione scrivi_parole(f):  
    parola = ""  
    per ogni carattere c di f + " ":  
        se c ≠ " " allora  
            parola = parola + c  
        altrimenti  
            scrivi parola  
            parola = ""
```

Lo spazio aggiunto in fondo (f + " ") serve a scrivere anche l'ultima parola: senza, «sole» non verrebbe mai scritta.

Con f = "il sole"

c	parola	Scrivi
i	"i"	
l	"il"	
␣	""	il
s	"s"	
o	"so"	
l	"sol"	
e	"sole"	
␣	""	sole



ESERCIZIO — LA PAROLA PIÙ LUNGA

Partite da `scrivi_parole` e scrivete lo pseudocodice di `parola_piu_lunga(f)`.

- ✘ Invece di scrivere ogni parola, confrontatela con la più lunga trovata fino a quel momento.
- ✘ Alla fine restituite la più lunga. È lo schema del massimo della lezione 2.
- ✘ Controllo: `parola_piu_lunga("il sole splende") = "splende"`,
`parola_piu_lunga("uno due") = "uno"`
- ✘ **Se due parole hanno la stessa lunghezza, quale restituisce la vostra funzione?**

SOLUZIONE — PAROLA PIÙ LUNGA

Pseudocodice

```
funzione parola_piu_lunga(f):  
    parola = ""  
    max = ""  
    per ogni carattere c di f + " ":  
        se c ≠ " " allora  
            parola = parola + c  
        altrimenti  
            se lunghezza(parola) > lunghezza(max) allora  
                max = parola  
            parola = ""  
    restituisci max
```

Lo spazio aggiunto in fondo (f + " ") fa da sentinella: garantisce che anche l'ultima parola venga confrontata. A parità di lunghezza resta la prima.

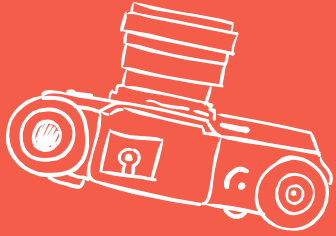
PAROLA PIÙ LUNGA IN PYTHON

```
def parola_piu_lunga(f):  
    parola = ""  
    max = ""  
    for c in f + " ":  
        if c != " ":  
            parola = parola + c  
        else:  
            if len(parola) > len(max):  
                max = parola  
            parola = ""  
    return max
```

«per ogni carattere *c* di *f* + " »
diventa *for c in f* + " »; *lunghezza*
diventa *len*.

**Il resto è lo stesso dello
pseudocodice, riga per riga.**

TRACCIARE LE POSIZIONI



TRE CARATTERI UGUALI

```
funzione tre_uguali(s):  
    i = 2  
    fintanto che i < lunghezza(s):  
        se s[i] = s[i-1] e  
           s[i] = s[i-2] allora  
            restituisci vero  
        i = i + 1  
    restituisci falso
```

Con s = "aabbbc"

a	a	b	b	b	c
0	1	2	3	4	5

$s[i-1]$ è il carattere prima di $s[i]$, $s[i-2]$ quello due posizioni prima. Si parte da $i = 2$ perché servono entrambi.

i	i < 6 ?	s[i-2] s[i-1] s[i]	Tre uguali?	Che cosa succede
2	Vero	a a b	no	i = 3
3	Vero	a b b	no	i = 4
4	Vero	b b b	sì	restituisci vero

Appena si trova una tripla la funzione risponde: gli altri caratteri non si guardano.

PROVATE VOI: "BASSOTTO"

Stessa funzione, con $s = \text{"Bassotto"}$. Completate la tabella e dite che cosa restituisce.

B	a	s	s	o	t	t	o
0	1	2	3	4	5	6	7

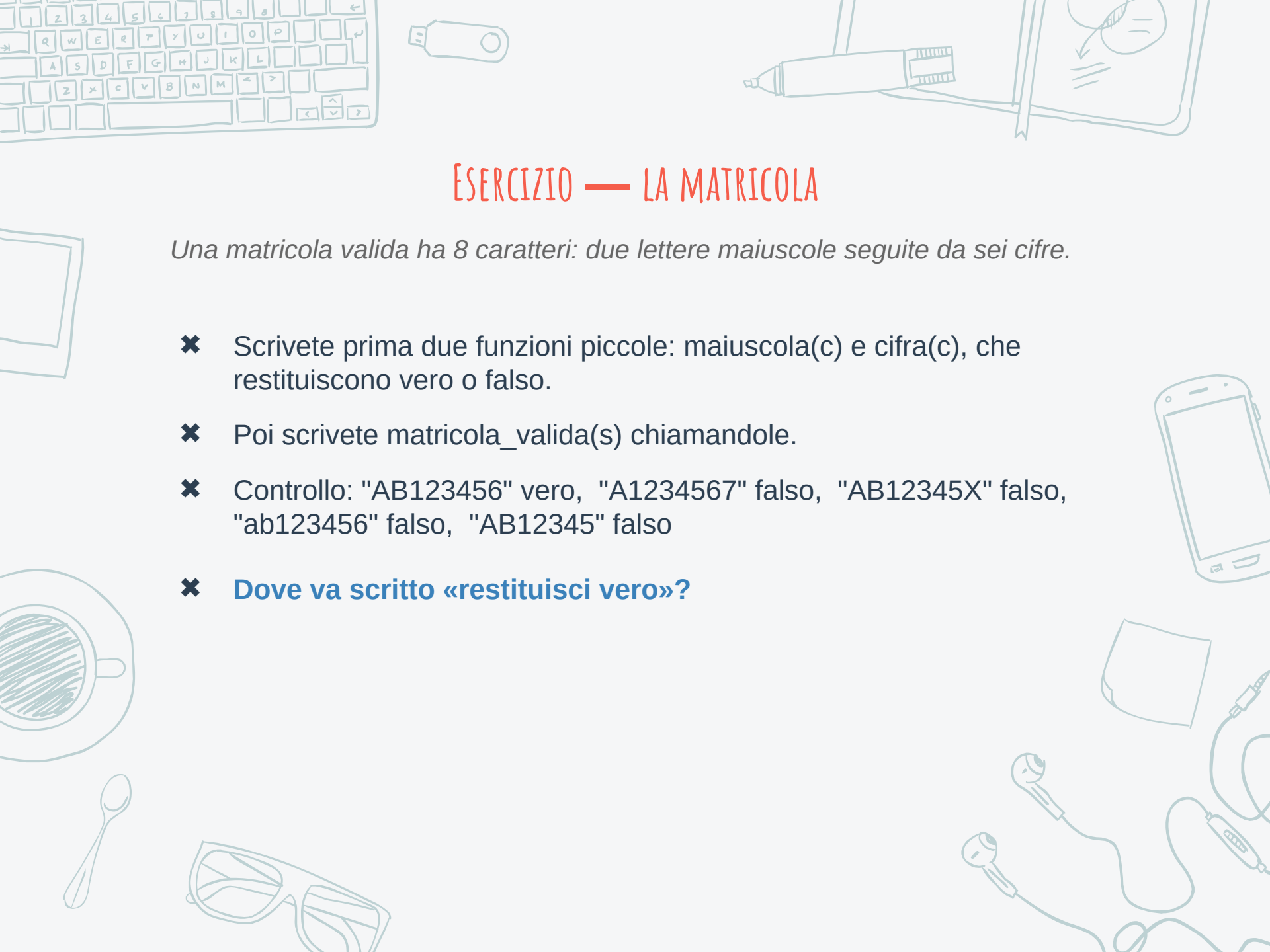
i	i < 8 ?	s[i-2] s[i-1] s[i]	Tre uguali?	Che cosa succede

Se finite prima: che cosa succederebbe partendo da $i = 0$?

SOLUZIONE — "BASSOTTO"

i	i < 8 ?	s[i-2] s[i-1] s[i]	Tre uguali?	Che cosa succede
2	Vero	B a s	no	i = 3
3	Vero	a s s	no	i = 4
4	Vero	s s o	no	i = 5
5	Vero	s o t	no	i = 6
6	Vero	o t t	no	i = 7
7	Vero	t t o	no	i = 8
8	Falso	—	—	esce dal ciclo, restituisci falso

Per rispondere «falso» la funzione ha dovuto guardare tutte le posizioni da 2 a 7. Per rispondere «vero» con "aabbbc" ne sono bastate tre.



ESERCIZIO — LA MATRICOLA

Una matricola valida ha 8 caratteri: due lettere maiuscole seguite da sei cifre.

- ✘ Scrivete prima due funzioni piccole: `maiuscola(c)` e `cifra(c)`, che restituiscono vero o falso.
- ✘ Poi scrivete `matricola_valida(s)` chiamandole.
- ✘ Controllo: "AB123456" vero, "A1234567" falso, "AB12345X" falso, "ab123456" falso, "AB12345" falso
- ✘ **Dove va scritto «restituisce vero»?**

SOLUZIONE — LA MATRICOLA

```
funzione maiuscola(c):  
    restituisci  $c \geq \text{"A"}$  e  $c \leq \text{"Z"}$ 
```

```
funzione cifra(c):  
    restituisci  $c \geq \text{"0"}$  e  $c \leq \text{"9"}$ 
```

```
funzione matricola_valida(s):  
    se  $\text{lunghezza}(s) \neq 8$  allora  
        restituisci falso  
     $i = 0$   
    fintanto che  $i < 8$ :  
        se  $i < 2$  e non  $\text{maiuscola}(s[i])$  allora  
            restituisci falso  
        se  $i \geq 2$  e non  $\text{cifra}(s[i])$  allora  
            restituisci falso  
         $i = i + 1$   
    restituisci vero
```

s	Esito	Si ferma
"AB123456"	vero	dopo il ciclo
"A1234567"	falso	$i = 1$
"AB12345X"	falso	$i = 7$
"ab123456"	falso	$i = 0$
"AB12345"	falso	lunghezza

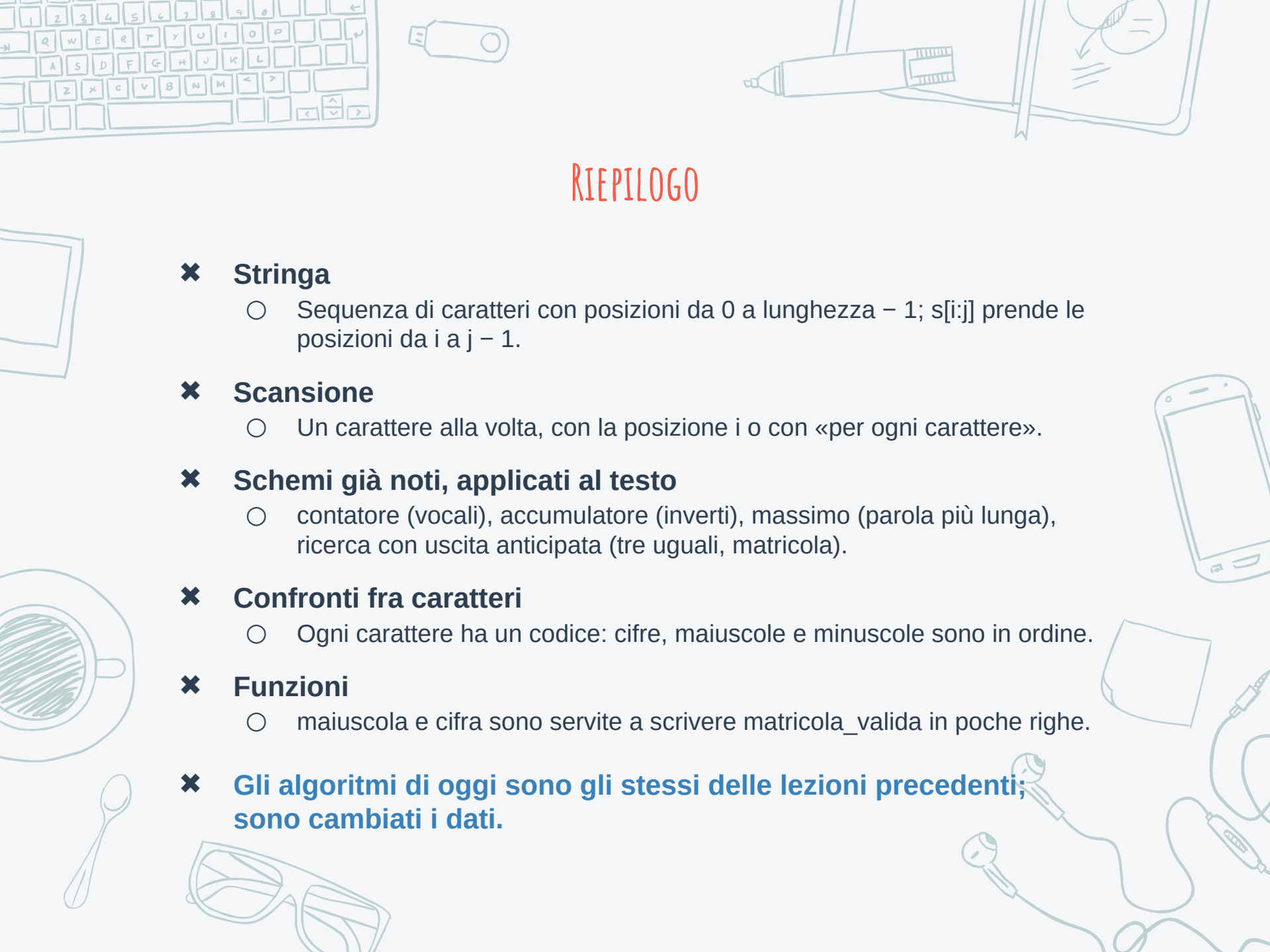
**maiuscola e cifra si possono
riusare in altri problemi, per
esempio nella sfida della
password.**

MATRICOLA IN PYTHON

```
def maiuscola(c):  
    return "A" <= c <= "Z"  
  
def cifra(c):  
    return "0" <= c <= "9"  
  
def matricola_valida(s):  
    if len(s) != 8:  
        return False  
    i = 0  
    while i < 8:  
        if i < 2 and not maiuscola(s[i]):  
            return False  
        if i >= 2 and not cifra(s[i]):  
            return False  
        i = i + 1  
    return True
```

In Python "A" <= c <= "Z" equivale a $c \geq "A"$ e $c \leq "Z"$.

Python ha funzioni già pronte per questi controlli (isupper, isdigit). Qui le abbiamo scritte noi per vedere come si fanno.



RIEPILOGO

✖ Stringa

- Sequenza di caratteri con posizioni da 0 a lunghezza - 1; $s[i:j]$ prende le posizioni da i a $j - 1$.

✖ Scansione

- Un carattere alla volta, con la posizione i o con «per ogni carattere».

✖ Schemi già noti, applicati al testo

- contatore (vocali), accumulatore (inverti), massimo (parola più lunga), ricerca con uscita anticipata (tre uguali, matricola).

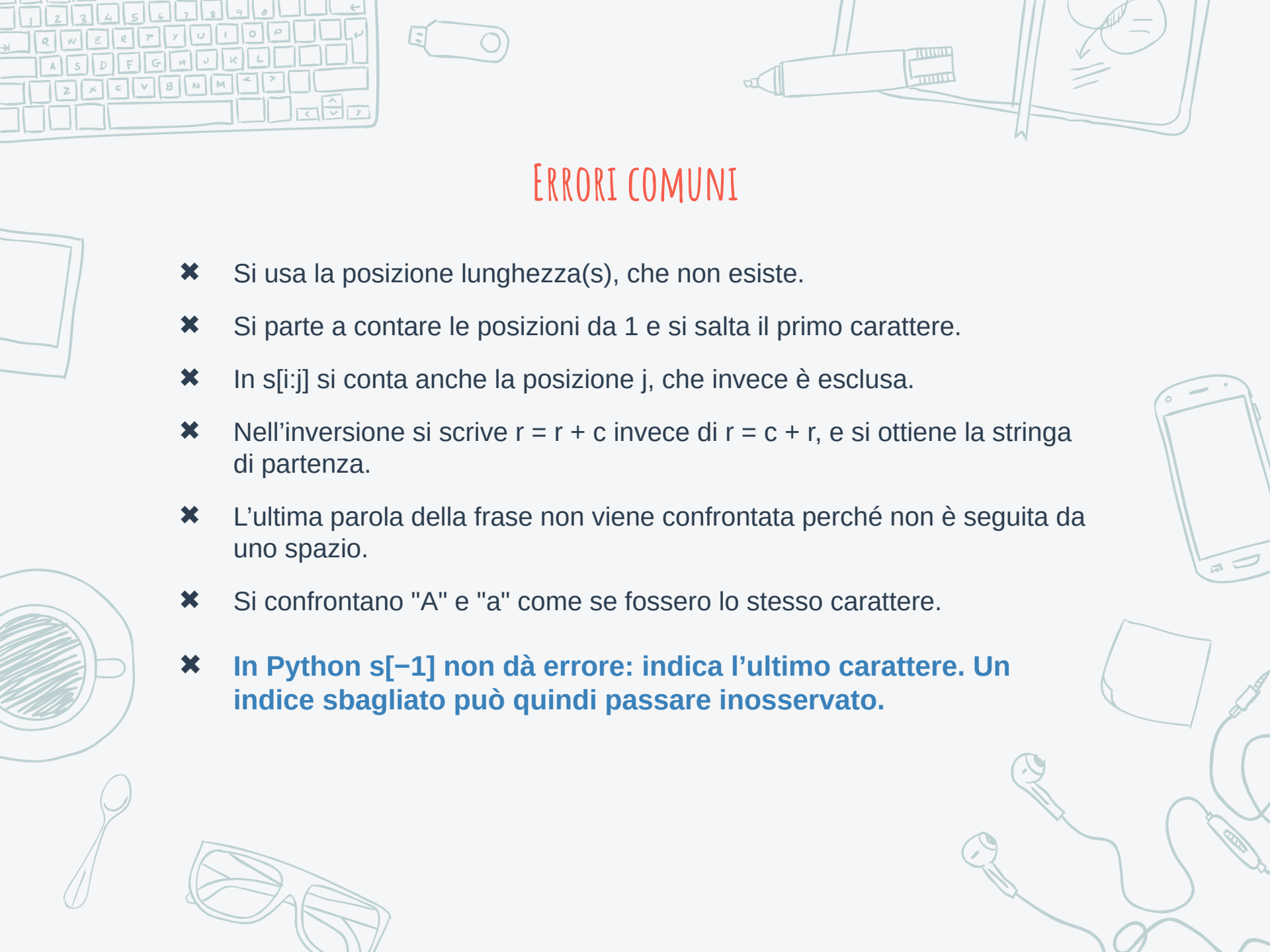
✖ Confronti fra caratteri

- Ogni carattere ha un codice: cifre, maiuscole e minuscole sono in ordine.

✖ Funzioni

- maiuscola e cifra sono servite a scrivere `matricola_valida` in poche righe.

✖ Gli algoritmi di oggi sono gli stessi delle lezioni precedenti; sono cambiati i dati.



ERRORI COMUNI

- ✗ Si usa la posizione lunghezza(s), che non esiste.
- ✗ Si parte a contare le posizioni da 1 e si salta il primo carattere.
- ✗ In `s[i:j]` si conta anche la posizione `j`, che invece è esclusa.
- ✗ Nell'inversione si scrive `r = r + c` invece di `r = c + r`, e si ottiene la stringa di partenza.
- ✗ L'ultima parola della frase non viene confrontata perché non è seguita da uno spazio.
- ✗ Si confrontano "A" e "a" come se fossero lo stesso carattere.
- ✗ **In Python `s[-1]` non dà errore: indica l'ultimo carattere. Un indice sbagliato può quindi passare inosservato.**

CINQUE MINUTI DI PYTHON

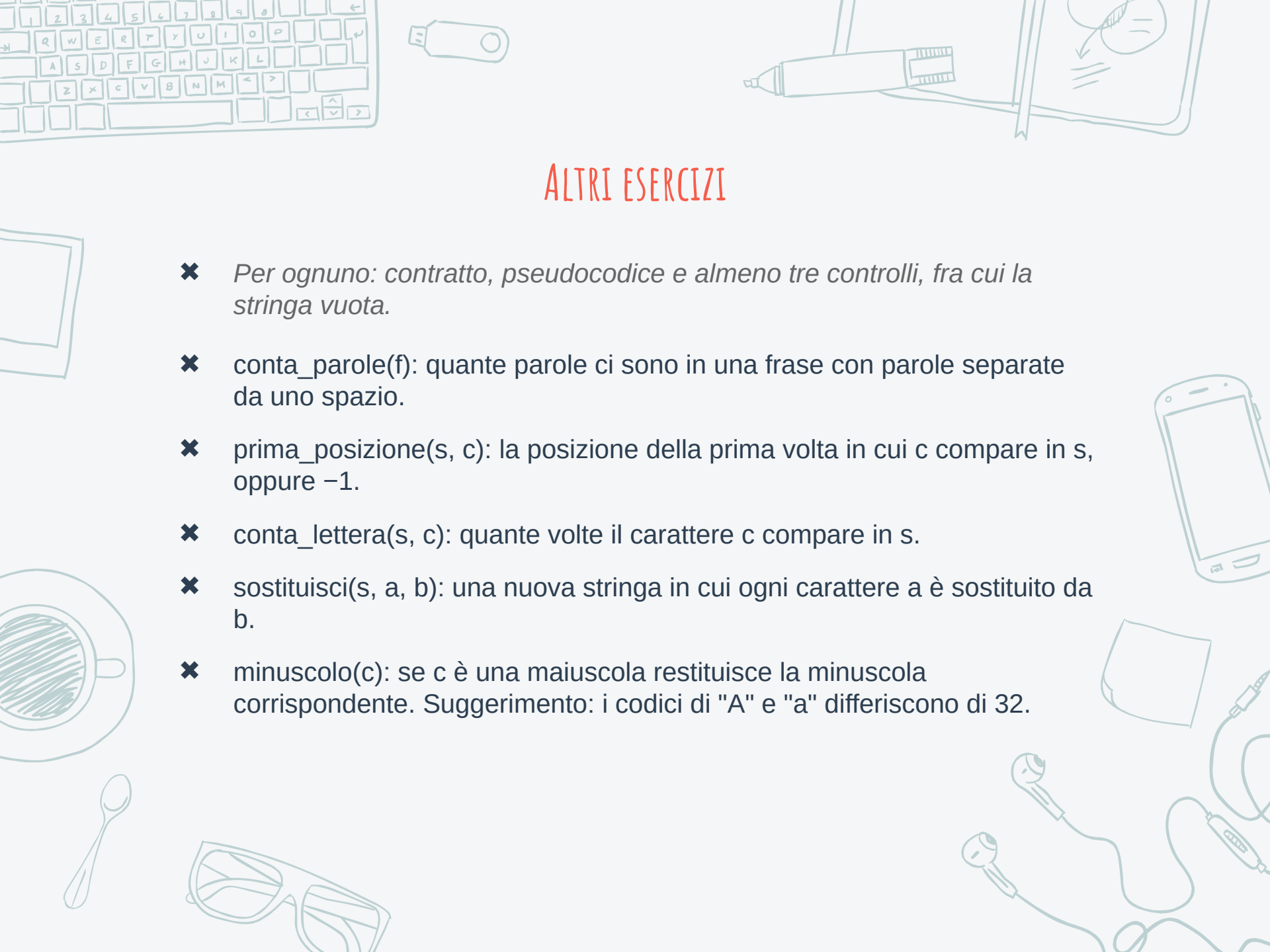
Le scritture di oggi e la loro traduzione in Python.

Pseudocodice	Python
lunghezza(s)	<code>len(s)</code>
s[i]	<code>s[i]</code>
per ogni carattere c di s:	<code>for c in s:</code>
c è in "aeiou"	<code>c in "aeiou"</code>
$c \geq "0"$ e $c \leq "9"$	<code>"0" <= c <= "9"</code>
$c \neq " "$	<code>c != " "</code>
s[i:j]	<code>s[i:j]</code>
r = c + r	<code>r = c + r</code>

Python offre anche funzioni già pronte, come `s.lower()` per le minuscole o `s[::-1]` per invertire. Si usano conoscendone il contratto; le vedremo nella lezione 7.

SFIDE FACOLTATIVE





ALTRI ESERCIZI

- ✘ *Per ognuno: contratto, pseudocodice e almeno tre controlli, fra cui la stringa vuota.*
- ✘ `conta_parole(f)`: quante parole ci sono in una frase con parole separate da uno spazio.
- ✘ `prima_posizione(s, c)`: la posizione della prima volta in cui `c` compare in `s`, oppure `-1`.
- ✘ `conta_lettera(s, c)`: quante volte il carattere `c` compare in `s`.
- ✘ `sostituisci(s, a, b)`: una nuova stringa in cui ogni carattere `a` è sostituito da `b`.
- ✘ `minuscolo(c)`: se `c` è una maiuscola restituisce la minuscola corrispondente. Suggerimento: i codici di "A" e "a" differiscono di 32.

ALTRI ESERCIZI — SOLUZIONI

Contatori, ricerca con uscita anticipata e riuso di conta_spazi.

conta_parole(f)

```
funzione conta_parole(f):  
    se f == "" allora  
        restituisci 0  
    restituisci conta_spazi(f) + 1
```

Con una parola sola non ci sono spazi; ogni spazio aggiunge una parola.

conta_lettera(s, c)

```
funzione conta_lettera(s, c):  
    n = 0  
    per ogni carattere x di s:  
        se x == c allora  
            n = n + 1  
    restituisci n
```

prima_posizione(s, c)

```
funzione prima_posizione(s, c):  
    i = 0  
    fintanto che i < lunghezza(s):  
        se s[i] == c allora  
            restituisci i  
        i = i + 1  
    restituisci -1
```

Controlli:

```
conta_parole("uno due tre") = 3  
conta_parole("") = 0  
prima_posizione("Roma", "m") = 2  
prima_posizione("Roma", "x") = -1  
conta_lettera("banana", "a") = 3
```

ALTRI ESERCIZI — SOLUZIONI (2)

Costruire una stringa nuova, un carattere alla volta.

sostituisci(s, a, b)

```
funzione sostituisci(s, a, b):  
    r = ""  
    per ogni carattere c di s:  
        se c = a allora  
            r = r + b  
        altrimenti  
            r = r + c  
    restituisci r
```

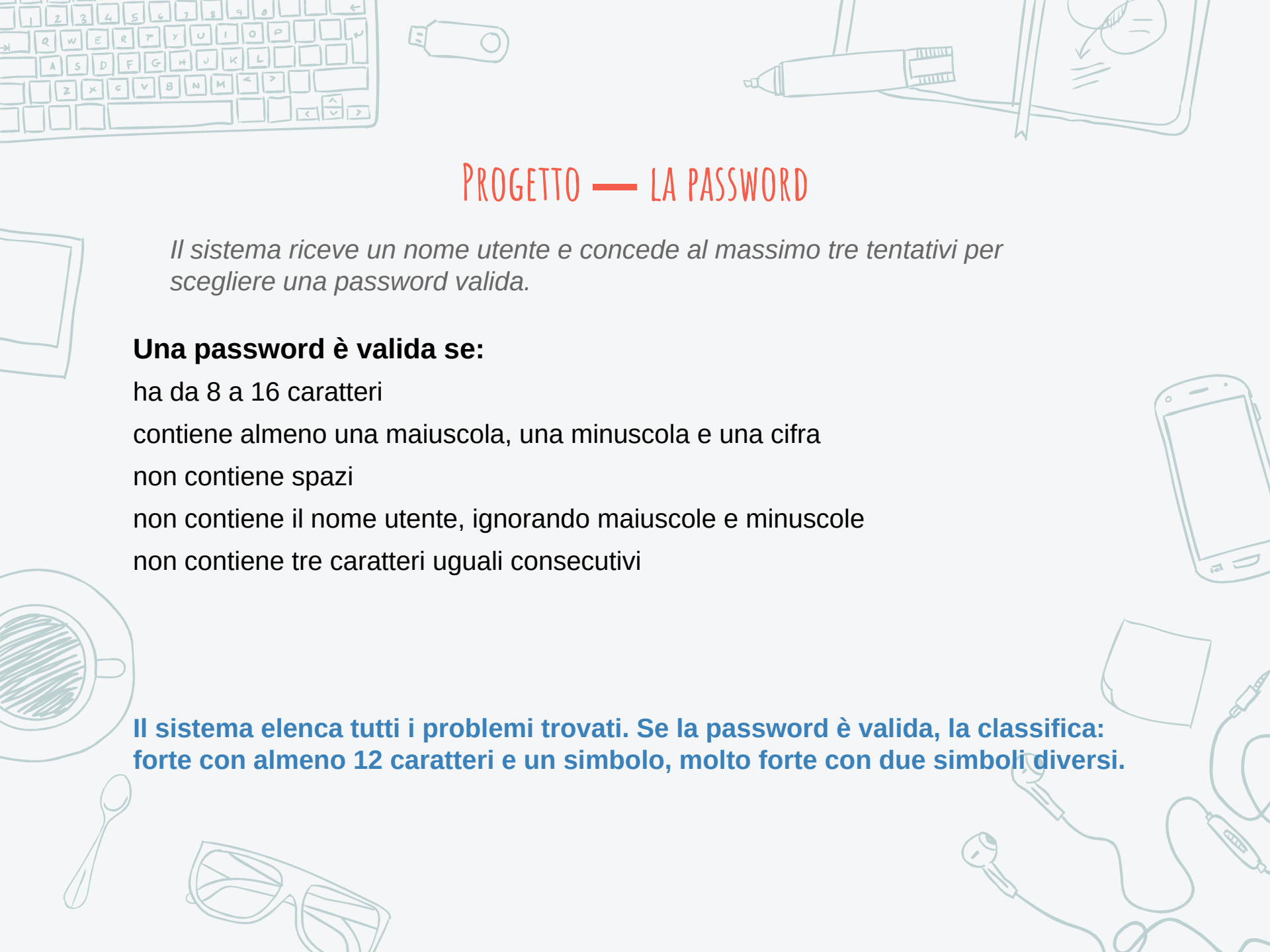
*È togli_spazi con un ramo altrimenti.
sostituisci("casa", "a", "o") = "coso".*

minuscolo(c)

```
funzione minuscolo(c):  
    M = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
    m = "abcdefghijklmnopqrstuvwxyz"  
    i = 0  
    fintanto che i < 26:  
        se M[i] = c allora  
            restituisci m[i]  
        i = i + 1  
    restituisci c
```

*Si cerca c fra le maiuscole e si restituisce la
lettera nella stessa posizione fra le minuscole.
Se c non è una maiuscola resta com'è.*

minuscolo serve anche nel progetto: per cercare il nome utente senza badare a maiuscole e minuscole.



PROGETTO — LA PASSWORD

Il sistema riceve un nome utente e concede al massimo tre tentativi per scegliere una password valida.

Una password è valida se:

ha da 8 a 16 caratteri

contiene almeno una maiuscola, una minuscola e una cifra

non contiene spazi

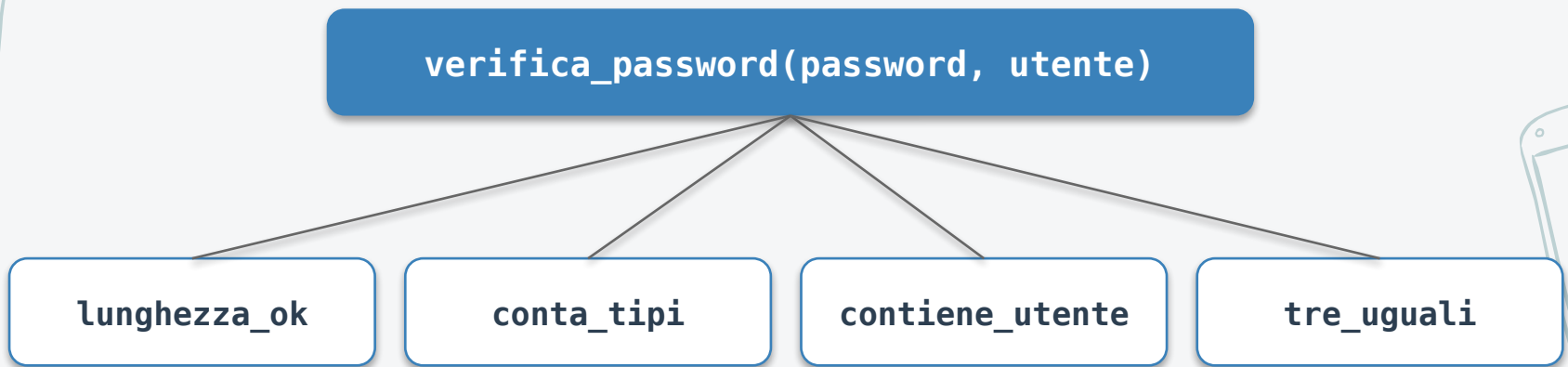
non contiene il nome utente, ignorando maiuscole e minuscole

non contiene tre caratteri uguali consecutivi

Il sistema elenca tutti i problemi trovati. Se la password è valida, la classifica: forte con almeno 12 caratteri e un simbolo, molto forte con due simboli diversi.

PASSWORD — SCOMPORRE

Prima di scrivere lo pseudocodice, dividete il problema in funzioni.



Per ogni funzione scrivete il contratto: che cosa riceve e che cosa restituisce. `conta_tipi` può restituire quante maiuscole, minuscole, cifre, spazi e simboli ci sono.

`tre_uguali` l'avete già tracciata a lezione; maiuscola e cifra vengono dalla matricola.

PASSWORD — CASI DI PROVA

Nome utente di prova: Marco

Password	Risultato atteso
abc	Troppo corta, senza maiuscola e senza cifra
Abcdef12	Valida
Statistica26!	Valida, forte
Marco2026!	Non valida: contiene il nome utente
Abc111def!	Non valida: tre caratteri uguali consecutivi
CorsoPython26!?	Valida, molto forte

Consegna: diagramma di flusso, pseudocodice e almeno sei casi di prova. Il Python è facoltativo.

PASSWORD — FUNZIONI PICCOLE

maiuscola e cifra vengono dalla matricola, tre_uguali dalla tabella di traccia.

Che tipo di carattere è?

```
funzione minuscola(c):  
    restituisci  $c \geq "a"$  e  $c \leq "z"$ 
```

```
funzione tipo(c):  
    se maiuscola(c) allora  
        restituisci "maiuscola"  
    se minuscola(c) allora  
        restituisci "minuscola"  
    se cifra(c) allora  
        restituisci "cifra"  
    se  $c = " "$  allora  
        restituisci "spazio"  
    restituisci "simbolo"
```

Contare

```
funzione conta_tipo(s, t):  
     $n = 0$   
    per ogni carattere c di s:  
        se tipo(c) = t allora  
             $n = n + 1$   
    restituisci n  
  
funzione simboli_diversi(s):  
     $r = ""$   
    per ogni carattere c di s:  
        se tipo(c) = "simbolo" e  
            non (c è in r) allora  
             $r = r + c$   
    restituisci lunghezza(r)
```

Una funzione non può restituire cinque numeri insieme: al posto di conta_tipi si scrive conta_tipo(s, t) e la si chiama una volta per ogni tipo.

PASSWORD — IL NOME UTENTE

«Marco2026!» contiene «marco» se si ignorano maiuscole e minuscole.

Tutto in minuscolo

```
funzione tutto_minuscolo(s):  
    r = ""  
    per ogni carattere c di s:  
        r = r + minuscolo(c)  
    restituisci r
```

```
funzione contiene_utente(p, u):  
    pm = tutto_minuscolo(p)  
    um = tutto_minuscolo(u)  
    restituisci contiene(pm, um)
```

Cercare un pezzo di testo

```
funzione contiene(s, t):  
    n = lunghezza(t)  
    i = 0  
    fintanto che  $i \leq \text{lunghezza}(s) - n$ :  
        se  $s[i : i + n] = t$  allora  
            restituisci vero  
        i = i + 1  
    restituisci falso
```

*contiene prova ogni posizione di partenza i e
confronta la sottostringa lunga quanto t.*

Con $p = \text{"Marco2026!"}$ e $u = \text{"Marco"}$: "marco2026!" contiene "marco" in posizione 0, quindi la risposta è vero.

PASSWORD — LA VERIFICA

verifica_password scrive tutti i problemi trovati e restituisce vero se non ce ne sono.

```
funzione verifica_password(p, u):  
    ok = vero  
    se lunghezza(p) < 8 allora  
        scrivi "troppo corta"; ok = falso  
    se lunghezza(p) > 16 allora  
        scrivi "troppo lunga"; ok = falso  
    se conta_tipo(p, "maiuscola") = 0 allora  
        scrivi "manca una maiuscola"; ok = falso  
    se conta_tipo(p, "minuscola") = 0 allora  
        scrivi "manca una minuscola"; ok = falso  
    se conta_tipo(p, "cifra") = 0 allora  
        scrivi "manca una cifra"; ok = falso  
    se conta_tipo(p, "spazio") > 0 allora  
        scrivi "contiene spazi"; ok = falso  
    se contiene_utente(p, u) allora  
        scrivi "contiene il nome utente"; ok = falso  
    se tre_uguali(p) allora  
        scrivi "tre caratteri uguali di fila"; ok = falso  
    restituisci ok
```

I controlli non si fermano al primo problema: ok diventa falso, ma si va avanti per poterli elencare tutti.

PASSWORD — FORZA E TENTATIVI

forza si chiama solo su una password valida. Il programma concede tre tentativi.

```
leggi u
tentativi = 0
accettata = falso
fintanto che tentativi < 3 e non accettata:
    leggi p
    tentativi = tentativi + 1
    se verifica_password(p, u) allora
        accettata = vero
        scrivi "accettata:", forza(p)
se non accettata allora
    scrivi "tentativi esauriti"
```

```
funzione forza(p):
    s = simboli_diversi(p)
    se lunghezza(p) ≥ 12 e s ≥ 2
        restituisci "molto forte"
    se lunghezza(p) ≥ 12 e s ≥ 1
        restituisci "forte"
    restituisci "valida"
```

Password (utente Marco)	Che cosa scrive
abc	troppo corta, manca una maiuscola, manca una cifra
Abcdef12	accettata: valida
Statistica26!	accettata: forte
Marco2026!	contiene il nome utente
Abc111def!	tre caratteri uguali di fila
CorsoPython26!?	accettata: molto forte

PASSWORD IN PYTHON

Senza metodi. maiuscola, cifra e tre_uguali sono quelle scritte a lezione.

```
def minuscola(c):  
    return "a" <= c <= "z"  
  
def tipo(c):  
    if maiuscola(c):  
        return "maiuscola"  
    if minuscola(c):  
        return "minuscola"  
    if cifra(c):  
        return "cifra"  
    if c == " ":  
        return "spazio"  
    return "simbolo"  
  
def conta_tipo(s, t):  
    n = 0  
    for c in s:  
        if tipo(c) == t:  
            n = n + 1  
    return n
```

```
def simboli_diversi(s):  
    r = ""  
    for c in s:  
        if tipo(c) == "simbolo":  
            if not c in r:  
                r = r + c  
    return len(r)  
  
def minuscolo(c):  
    M = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
    m = "abcdefghijklmnopqrstuvwxyz"  
    i = 0  
    while i < 26:  
        if M[i] == c:  
            return m[i]  
        i = i + 1  
    return c
```

PASSWORD IN PYTHON (2)

Il nome utente, la forza e il programma principale. input() legge una riga scritta dall'utente.

```
def tutto_minuscolo(s):
    r = ""
    for c in s:
        r = r + minuscolo(c)
    return r

def contiene(s, t):
    n = len(t)
    i = 0
    while i <= len(s) - n:
        if s[i:i + n] == t:
            return True
        i = i + 1
    return False

def contiene_utente(p, u):
    pm = tutto_minuscolo(p)
    um = tutto_minuscolo(u)
    return contiene(pm, um)
```

```
def forza(p):
    s = simboli_diversi(p)
    if len(p) >= 12 and s >= 2:
        return "molto forte"
    if len(p) >= 12 and s >= 1:
        return "forte"
    return "valida"

u = input("nome utente: ")
tentativi = 0
accettata = False
while tentativi < 3 and not accettata:
    p = input("password: ")
    tentativi = tentativi + 1
    if verifica_password(p, u):
        accettata = True
        print("accettata:", forza(p))
if not accettata:
    print("tentativi esauriti")
```

PASSWORD IN PYTHON (3)

La verifica. Il punto e virgola mette due istruzioni sulla stessa riga.

```
def verifica_password(p, u):  
    ok = True  
    if len(p) < 8:  
        print("troppo corta"); ok = False  
    if len(p) > 16:  
        print("troppo lunga"); ok = False  
    if conta_tipo(p, "maiuscola") == 0:  
        print("manca una maiuscola"); ok = False  
    if conta_tipo(p, "minuscola") == 0:  
        print("manca una minuscola"); ok = False  
    if conta_tipo(p, "cifra") == 0:  
        print("manca una cifra"); ok = False  
    if conta_tipo(p, "spazio") > 0:  
        print("contiene spazi"); ok = False  
    if contiene_utente(p, u):  
        print("contiene il nome utente"); ok = False  
    if tre_uguali(p):  
        print("tre caratteri uguali di fila"); ok = False  
    return ok
```

Con i metodi già pronti di Python (`p.lower()`, `c.isupper()`, `c.isdigit()`) il programma si accorcia, ma fa le stesse cose.