

SONY®

MSX

Introduzione all'
MSX-BASIC



HIT BIT

Introduzione all'
MSX-BASIC

INDICE

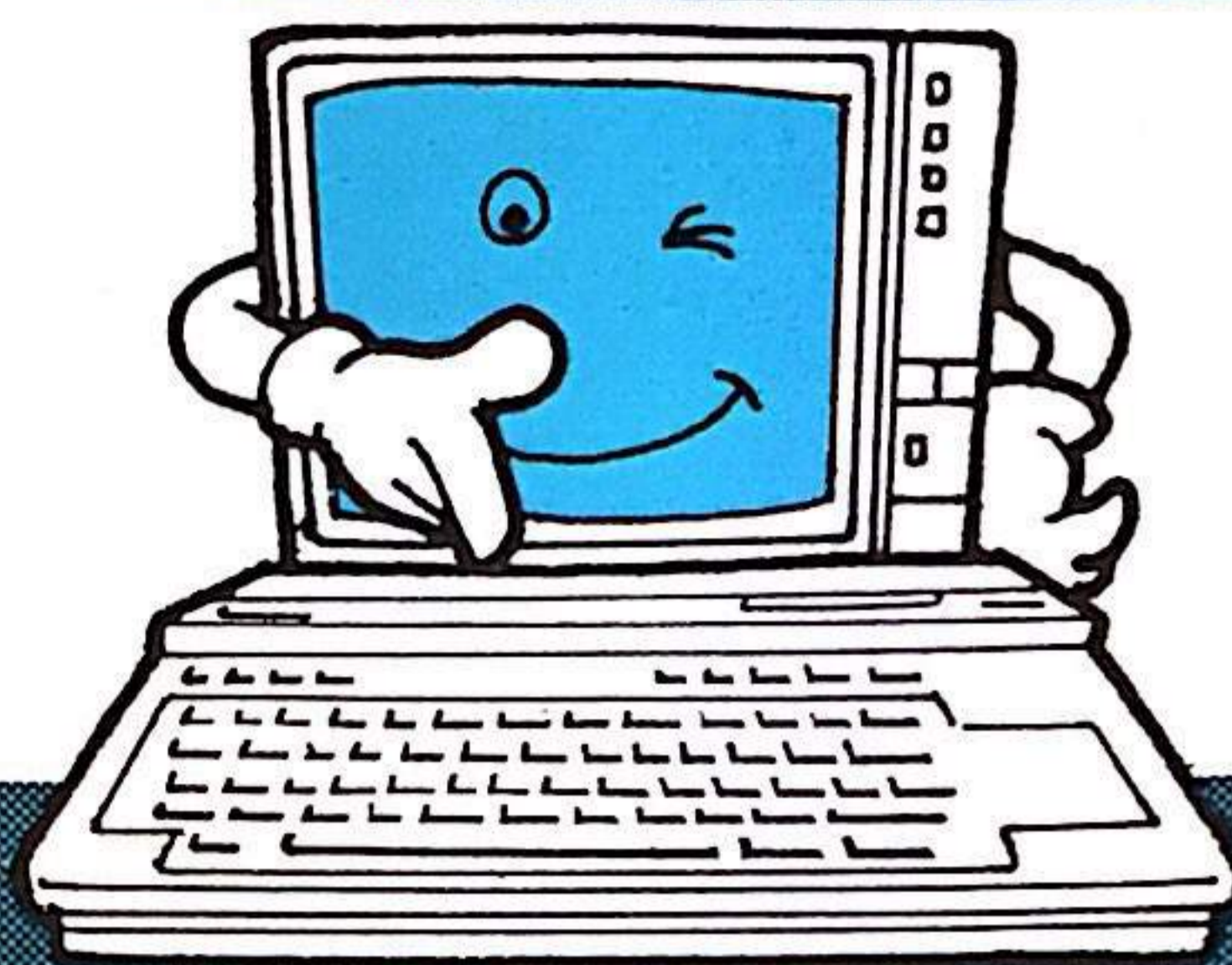
Per cominciare	4
Alcune cose a proposito di questo libro	4
Fido, il cane intelligente.....	5
Il vostro computer Sony	6
Mettiamo al lavoro il nostro computer	7
Comandi e immissioni.....	7
Alcuni giochetti che Fido non sa fare	10
Stampiamo la risposta.....	14
Numeri, lettere, variabili	17
Numeri o lettere: sono la stessa cosa.....	18
Una lettera diventa una variabile	19
Fabbrichiamo il nostro primo programma	27
Progettiamo un programma.....	28
Scriviamo un programma:	
semplice come uno, due, tre	31
Correggiamo il programma: cerchiamo gli errori	32
Facciamo girare il programma	34
Grafici: stelle luminose	37
Un programma di grafici	37
Numeri casuali.....	42
Mettiamo del colore nei nostri programmi.....	46
Conserviamo i nostri programmi	52
Collegamento del registratore.....	53
Facciamo parlare il computer	54
Siamo sicuri che il nastro ha registrato il programma?.....	55
Carichiamo un programma	56

Decisioni e giochi	58
Azzecato o sbagliato—decide il computer	60
Semplifichiamo il programma	61
Grafici che si muovono	64
Visualizziamo, cancelliamo, visualizziamo, cancelliamo	67
Mettiamo tutto insieme	71
Aggiungiamo colore e suono	71
Gioco dell'indovina il numero + suono + colore	73
Giochiamo con la macchinetta mangiasoldi	77
Cos'è una variabile insieme?	78
Fabbrichiamo una macchinetta mangiasoldi	79
Cos'è una variabile di stringa	85
Come fermare un ciclo con INKEY	86
Qual è il punteggio?	90
Gli ultimi ritocchi	95
Per facilitare la lettura dei programmi	98
Mettiamo il titolo al programma	99
Ne abbiamo fatta di strada!	102
Esercitiamoci con i comandi BASIC	103
PRINT	103
INPUT	106
FOR—NEXT	108
IF—THEN e IF—THEN—ELSE	111
DIM (Variabili insieme)	112
Indice analitico	117

Questo manuale è destinato solo all'uso del sistema Sony MSX.

PER COMINCIARE

- A proposito di questo libro
- Fido, il cane intelligente
- Il vostro computer Sony



A PROPOSITO DI QUESTO LIBRO

Questo libro vi insegnerà il BASIC.

Che cos'è il BASIC? È un linguaggio facile che vi permetterà di "parlare" con il computer. Questo linguaggio si chiama BASIC perché è il modo più semplice, **fondamentale** per parlare con un computer.

Ma perché vogliamo parlare con un computer?—direte. Perché un computer, il vostro computer Sony, vi può aiutare a fare un'infinità di cose—a giocare, a risolvere problemi, a fare esercizi di matematica, a memorizzare informazioni e tante altre cose. Questo libro vi insegnerà, in un'ora o due, a divertirvi con il vostro computer Sony.

Innanzitutto impareremo a parlare al computer e a capire quello che ci dice (nel linguaggio BASIC). In seguito, useremo questo linguaggio speciale per far fare al computer dei giochetti interessanti—delle cose facili all'inizio, più difficili in seguito. Prima della fine di questo libro, sarete già capaci di usare il vostro computer per programmare il gioco di una macchinetta mangiasoldi, una di quelle che si vedono nelle sale gioco di Las Vegas.

Quando, dopo aver letto questo libro, il BASIC vi sarà divenuto più familiare, potrete leggere e usare i programmi delle riviste di computer e giocare con gli amici che, come voi, possiedono un computer. Se poi voleste imparare qualcosa di più sul BASIC, non dovete far altro che leggere uno dei tanti libri reperibili in qualsiasi libreria, più lunghi e complessi.

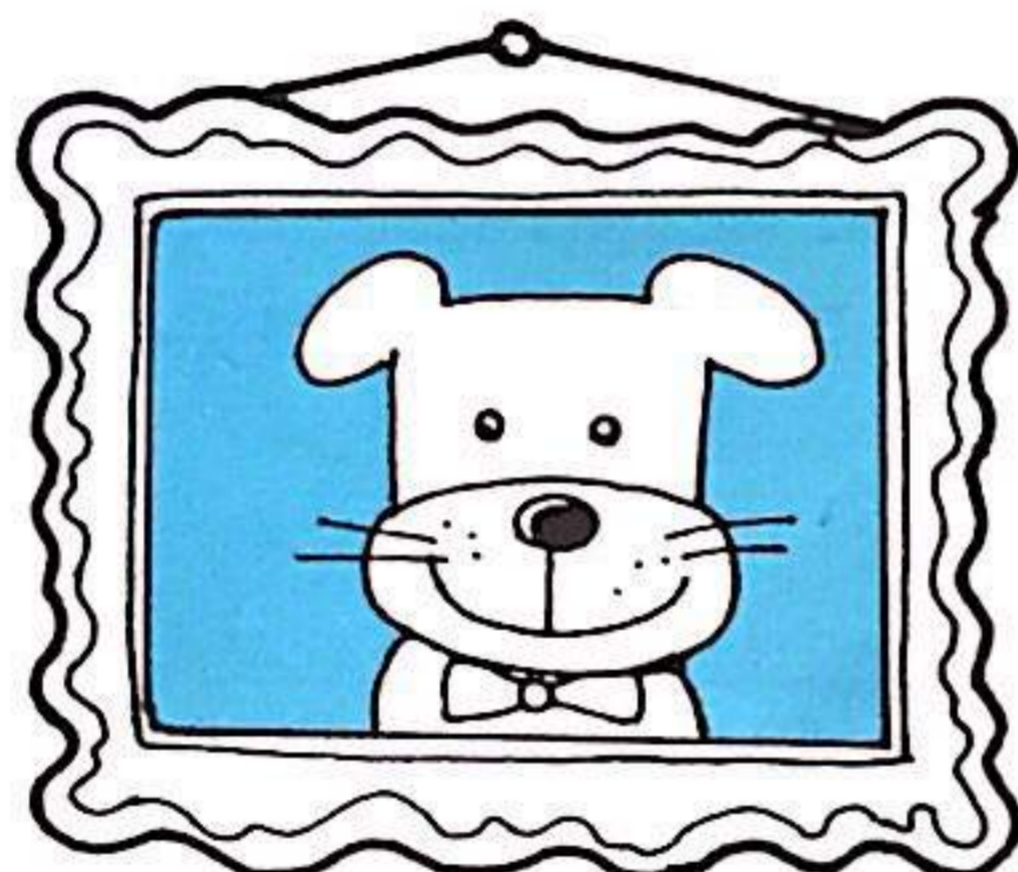
Per parlare di computer e di programmi di computer, useremo tanti termini speciali e comandi, tutti elencati in un **indice** alla fine di questo libro. Se volete controllare il significato di una parola o di un comando, basta che consultiate l'indice e lo saprete subito.

Infine, c'è un'ultima cosa importante che dovete tenere in mente: i computer non si arrabbiano proprio mai. Sono macchine semplici e **simpatiche** a cui non interessa quanti errori facciamo mentre impariamo ad usarli. Non succede niente al vostro computer se vi capita di premere il tasto "sbagliato"—o se per caso premete un tasto qualsiasi. Perciò, niente paura. Solo, come succede sempre quando si fa una cosa per la prima volta, è naturale fare e quindi correggere degli errori, o tornare indietro per rifare cose già fatte e farle meglio. Non c'è problema: il computer Sony non perderà mai la pazienza.

In effetti, l'unica cosa importante da tenere in mente prima di leggere questo libro, è che nel computer Sony avete trovato un nuovo amico, un amico che aspetta solo di essere messo alla prova.

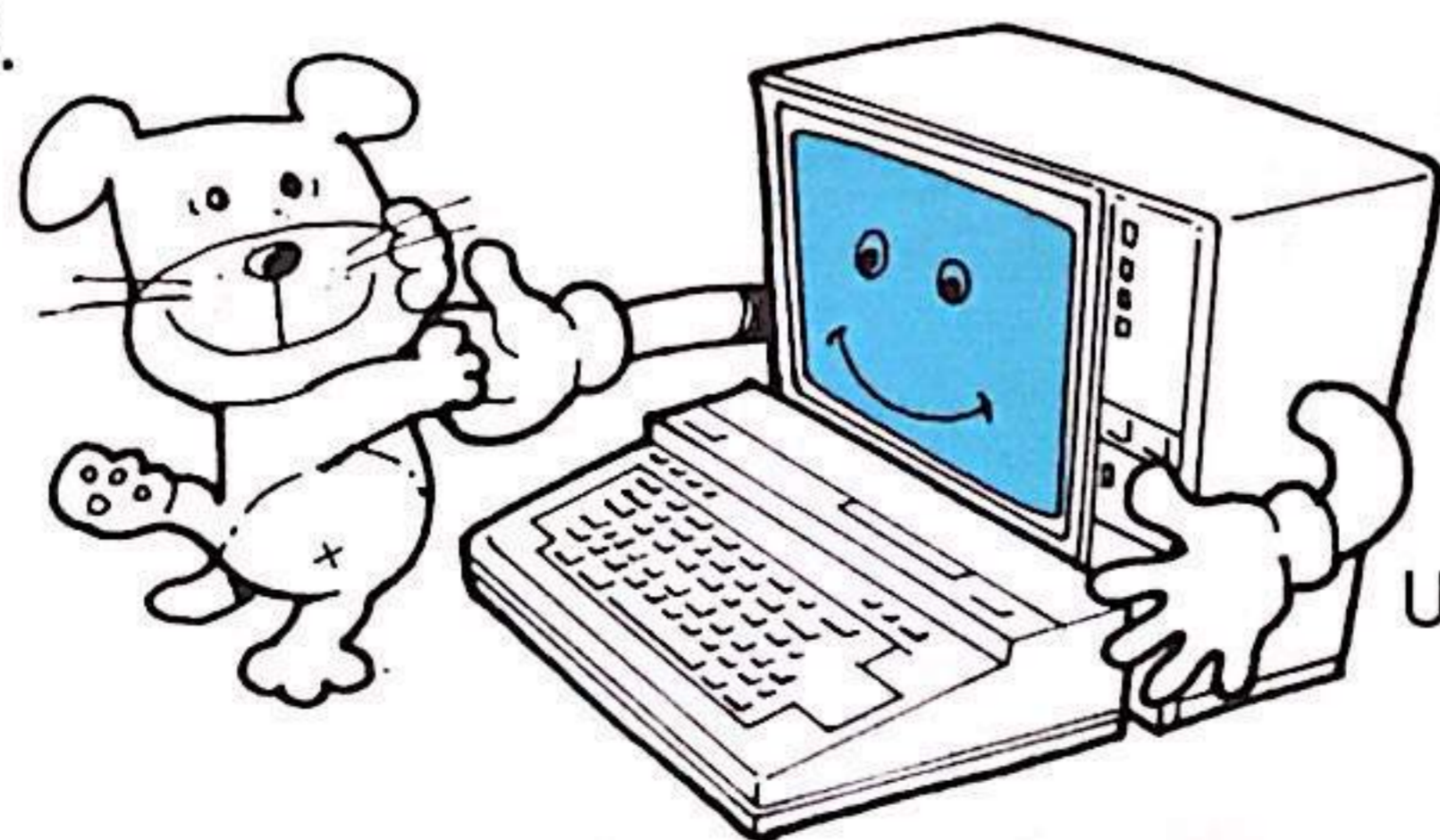
FIDO, IL CANE INTELLIGENTE

Un amico vi farà compagnia mentre incominciate a imparare il BASIC. È Fido, il nostro cane. Come la maggior parte dei cani, Fido è simpatico e fedele. Fido è un cane qualsiasi ma è piuttosto sveglio. Sa fare tantissime cose, ogni volta che gli vengono ordinate.



Fido

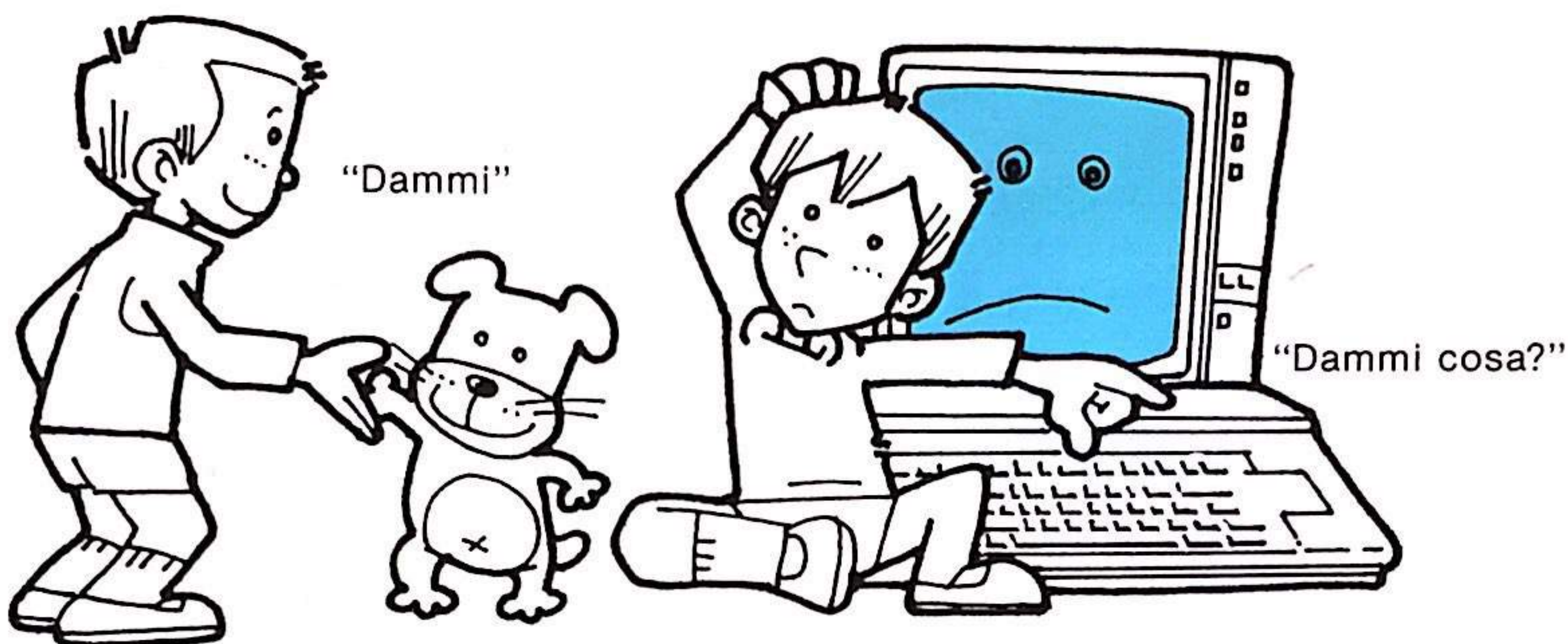
Perché mai Fido si trova in un libro sul BASIC e sui computer? Ci si trova perché i cani sono particolarmente bravi a obbedire ai **comandi** e fare quello che gli viene chiesto. Non solo, ma Fido e il nostro Computer Sony sono entrambi molto simpatici. Fra poco vedremo che anche il nostro computer esegue dei **comandi** ed è capace di fare dei giochetti, ma in modo diverso da Fido. Se pensiamo a come Fido e il nostro computer fanno le cose, ci sarà più facile capire come il computer ascolta, parla e pensa in BASIC.



Una coppia simpatica

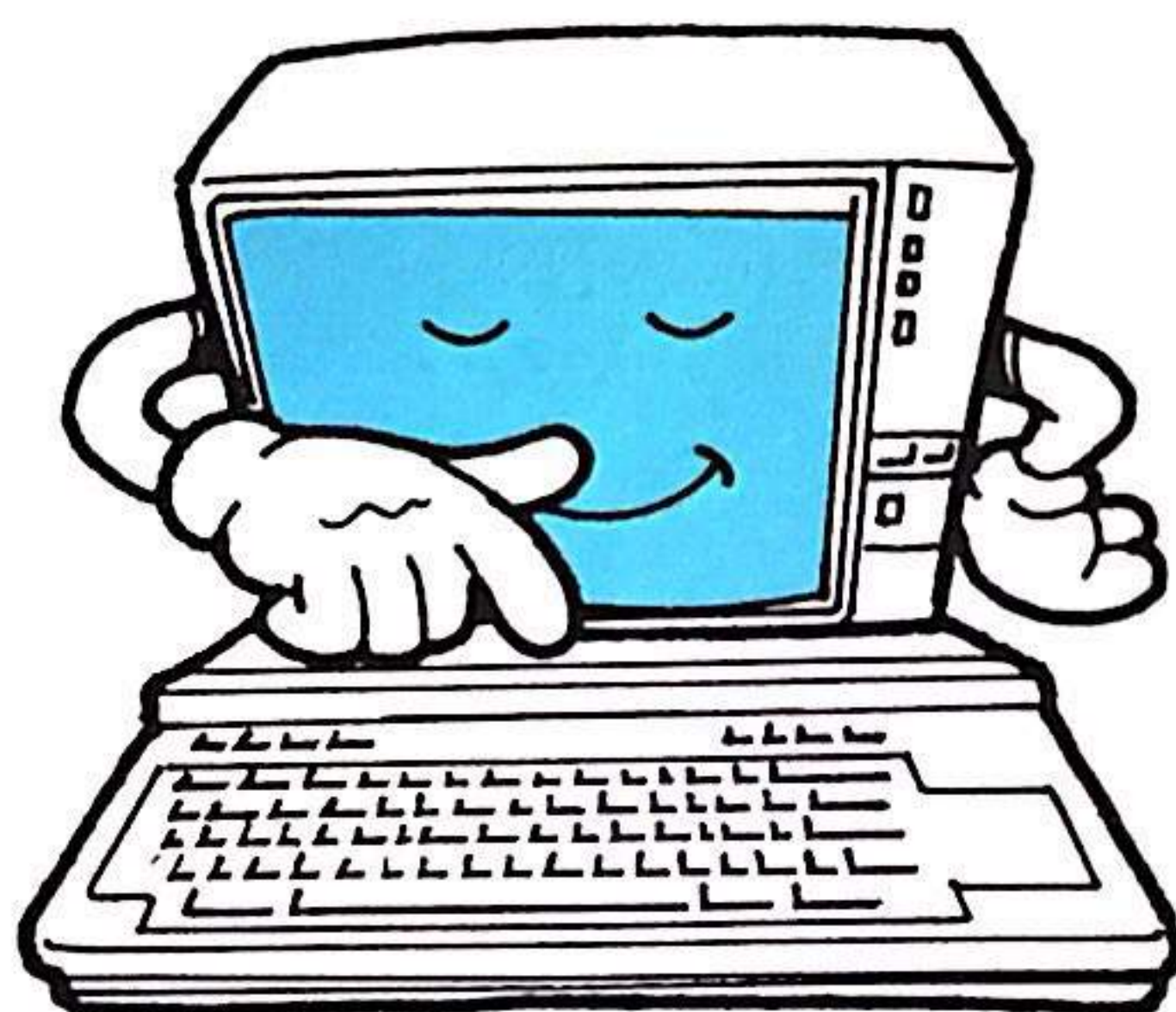
IL VOSTRO COMPUTER SONY

Naturalmente, Fido è bravo a fare le cose che gli vengono chieste. "Seduto", "Fa' una capriola", "Cercalo", "Portamelo"... tutto questo riesce facile a Fido. E al nostro computer? Proviamo con "dammi la mano".



Fido ha capito e ci ha dato la zampa, ma il computer non ha mani. Non ha nemmeno orecchi!

Come si può dire allora a un computer di fare qualcosa? L'"orecchio" di un computer—il suo modo di ascoltare i nostri comandi—è la **tastiera**.



Parliamo con le dita.

Prima di imparare il BASIC e prima di passare alla pagina seguente, leggete **Il manuale delle istruzioni** del vostro Computer Sony per conoscere la tastiera. Non è necessario imparare a memoria tutti i tasti. È molto importante però conoscere il **cursore**—capire che cos'è e come si muove. Vi conviene anche provare a scrivere sullo schermo delle **lettere**, dei **numeri** e dei **grafici**.

METTIAMO AL LAVORO IL NOSTRO COMPUTER

Come...

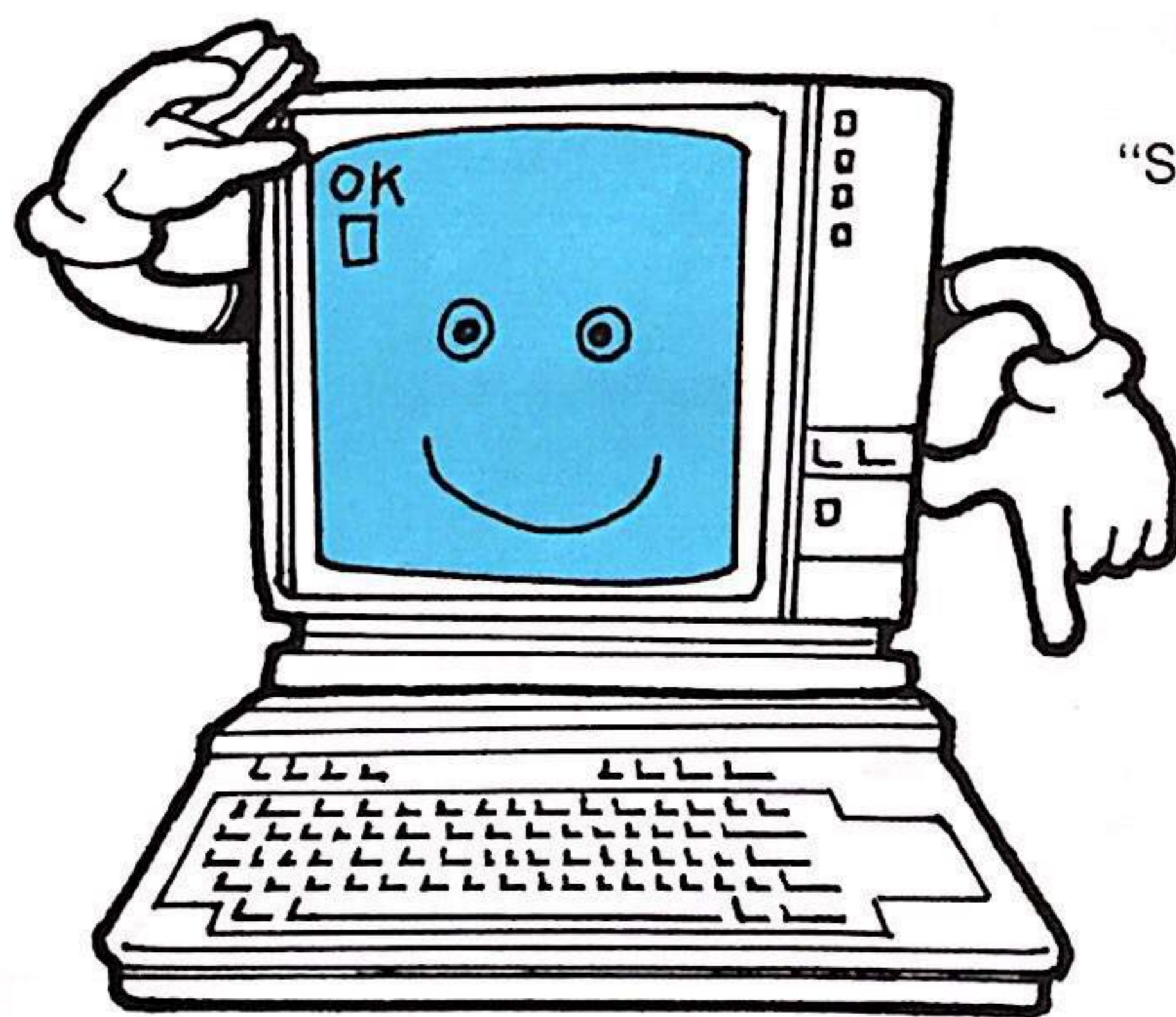
- Fermare il computer
- Dare comandi
- Scrivere a colori
- Usare il tasto delle maiuscole (Shift)
- Far fare l'aritmetica al computer



Ora che avete letto il manuale delle istruzioni e che conoscete la tastiera, proviamo a vedere che cosa sa fare il nostro computer Sony.

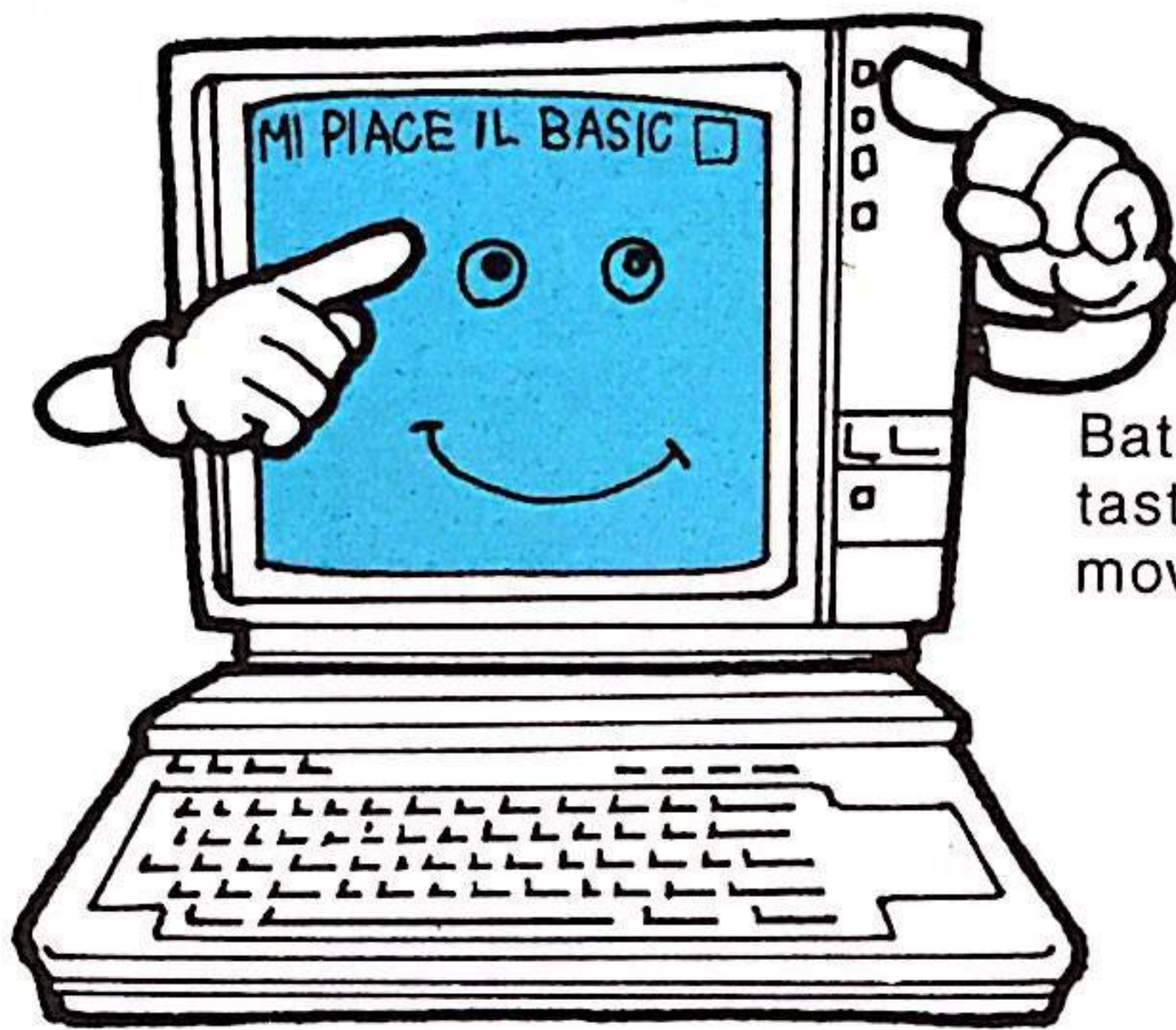
COMANDI E IMMISSIONI

Como abbiamo detto poco fa, la tastiera è l'“orecchio” per mezzo del quale il computer ascolta i nostri messaggi. Però, come facciamo a sapere che il computer ci sta ascoltando? Quando è pronto, ce lo dice con un amichevole “Ok”.



“Sempre pronto!”

Se usiamo la tastiera, sullo schermo appaiono le lettere e i numeri che battiamo. Il cursore si trova in mezzo allo schermo—e il messaggio “Ok” è rimasto indietro.

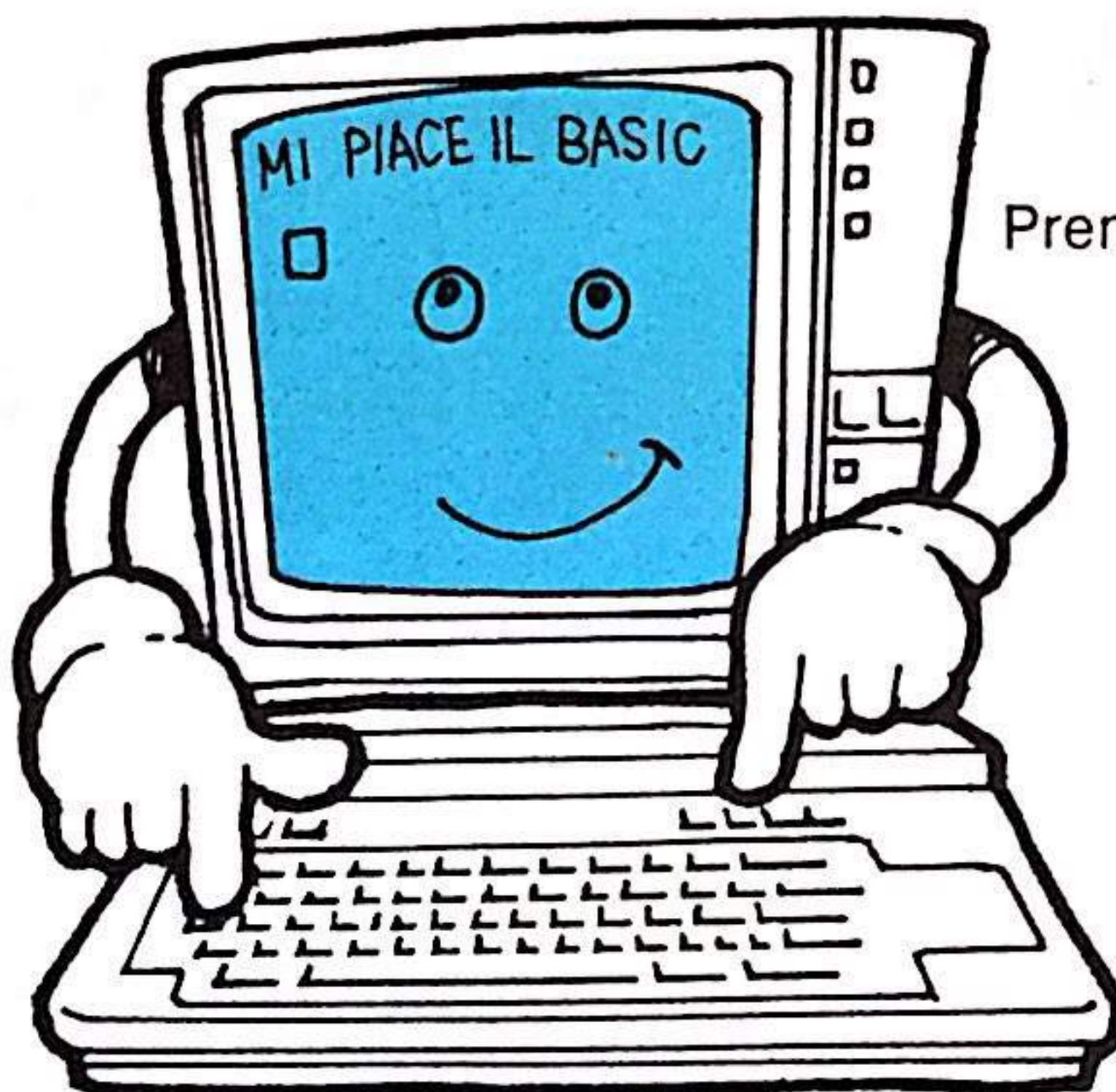


Battiamo alcune parole sulla tastiera e guardiamo il movimento del cursore.

Ora dobbiamo attirare l'attenzione del computer. Premiamo il tasto **CTRL** con un dito e il tasto **STOP** con un altro. Se i tasti **CTRL** e **STOP** vengono **premuti insieme**, il computer smette di ascoltare i vecchi comandi e si dedica completamente a quelli nuovi.

I comandi sono diversi dalle **immissioni**. Un'immissione è formata dalle lettere e dai numeri che costituiscono l'informazione che introduciamo nel computer—tutte le lettere che occorrono per far apparire sullo schermo "Mi piace il BASIC", ad esempio. Un'immissione non dice direttamente al computer di eseguire qualcosa. Per poter dire al computer che cosa fare con la nostra immissione, ci servono dei **comandi**, che vengono dati battendo certi tasti o certe parole. In un calcolatore tascabile, ad esempio, nella somma $2+2=4$, "2", "+" e "2" sono le immissioni. Il tasto "=" è invece un **comando** che dice al calcolatore di aggiungere 2 a 2 e di darci il risultato o emissione: 4.

Il computer funziona in questo modo. Un'altra cosa importante: la parola "Ok" appare **sempre** sopra il cursore quando un **comando** è stato immesso correttamente: ciò significa che quello che abbiamo fatto è OK.



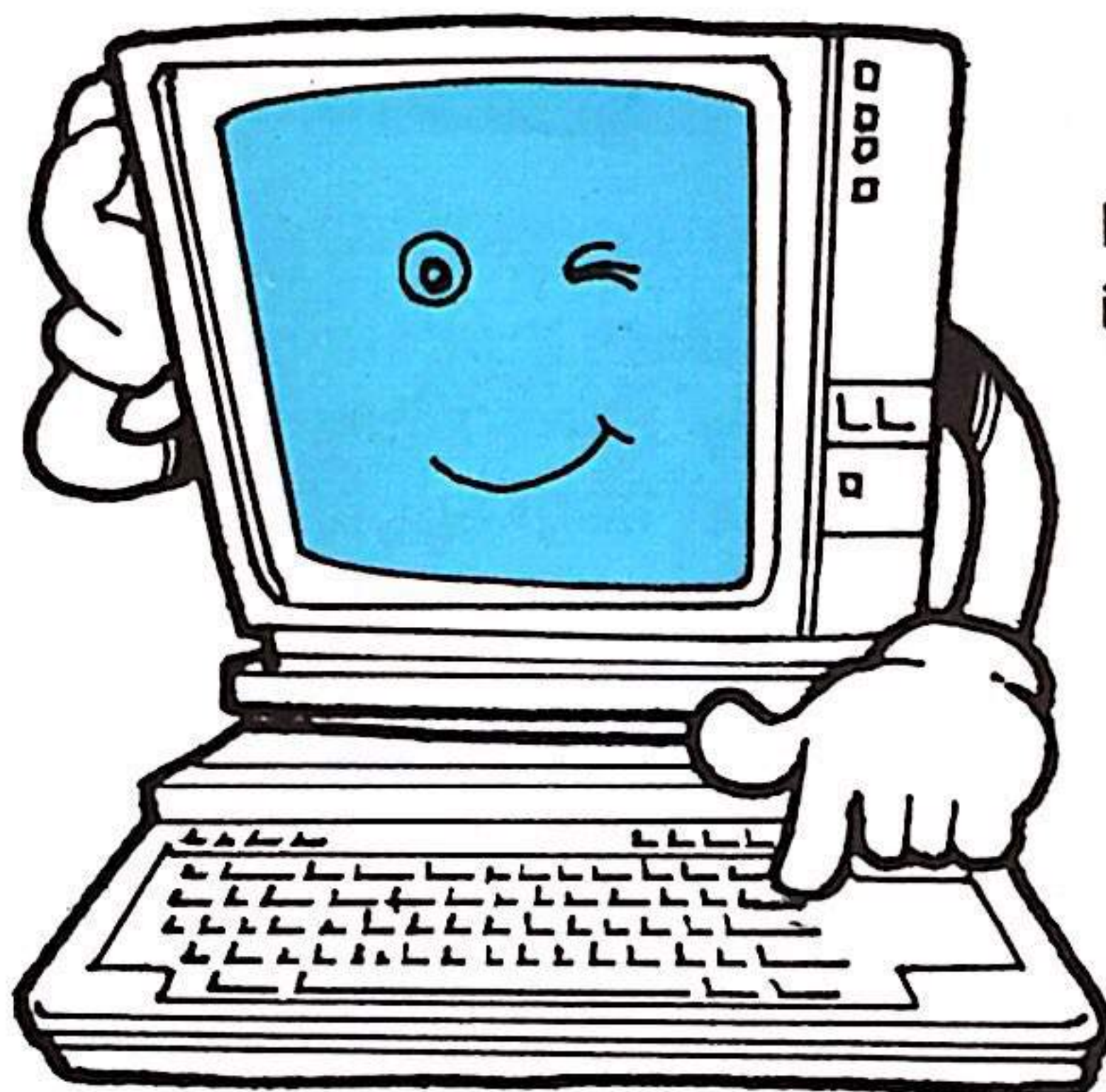
Premiamo **CTRL** e **STOP**.

Ora il computer è pronto a ricevere i nostri comandi. I comandi possono essere scritti in lettere minuscole o maiuscole—non fa differenza. Diciamogli di darci la mano battendo sulla tastiera “DAMMI LA MANO”.

DAMMI LA MANO

Non c'è alcuna reazione ancora.

Questo perché dobbiamo dire al computer quando eseguire il comando. Il tasto per fare questo si trova a destra ed è quello con su scritto “RETURN”. Ogni volta che diamo un comando al computer glielo faremo eseguire con il tasto `RETURN`¹⁾. Si usa spesso e per questo è un po' più grande degli altri tasti.



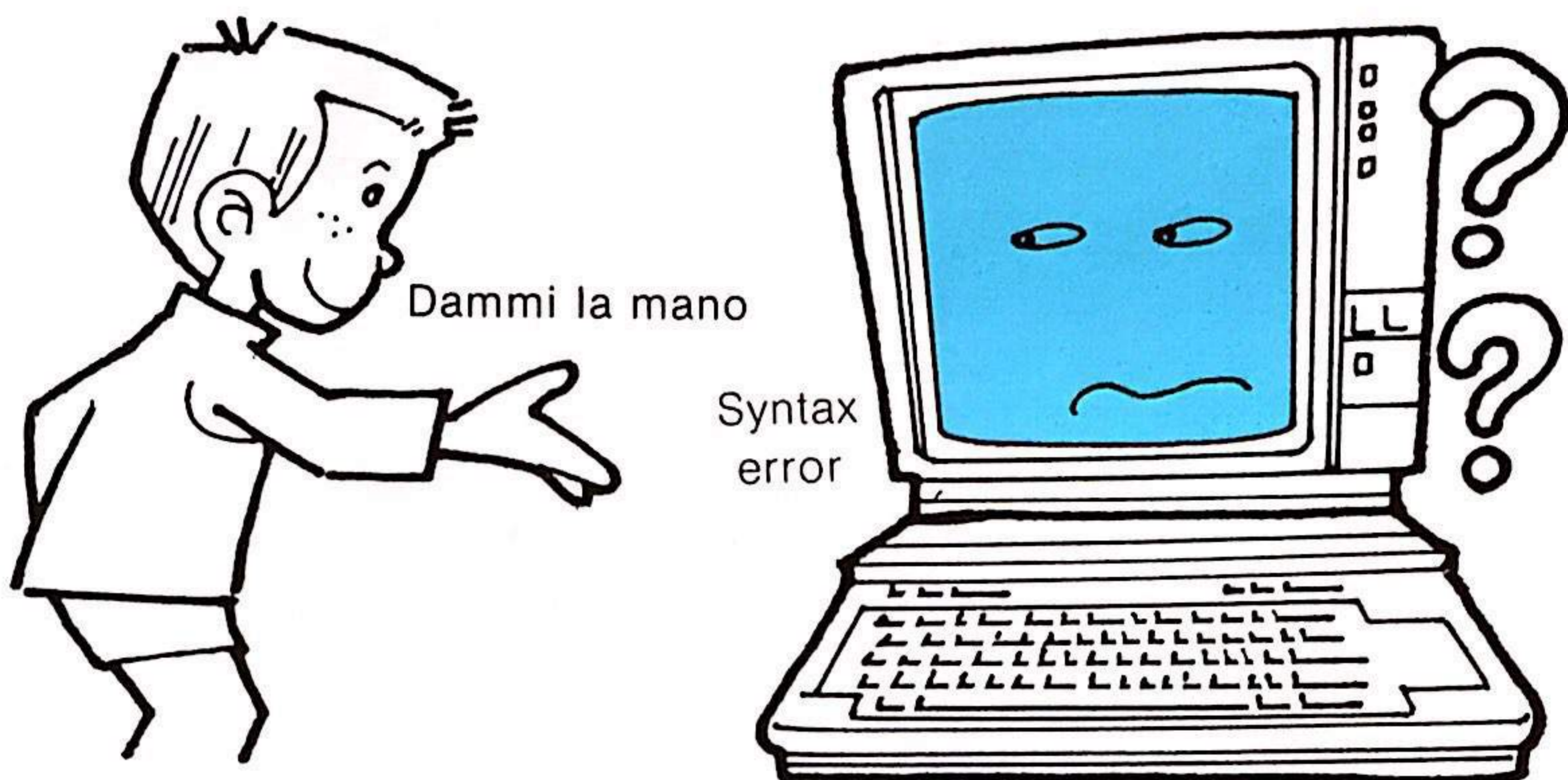
Il tasto `RETURN` dà i comandi.

Premiamo `RETURN` ed ecco la risposta. Non una stretta di mano, comunque, ma un “piii!”—ed una risposta di questo tipo:



Questo è un **messaggio d'errore**. Significa che il computer ha sentito il nostro comando—ma non lo capisce. La cosa è più che naturale in questo caso, perché i computer di solito non sanno dare la mano alla gente. Questo messaggio, “Syntax error”, significa che c'è un errore nel linguaggio usato, il linguaggio BASIC, naturalmente.

1) Il tasto  di alcune tastiere è uguale al tasto `RETURN` di questo manuale.



Fido, il nostro cane, capisce "Seduto" e "Qua la zampa" e tante altre parole, ma solo perché le ha imparate da qualcuno. Il nostro computer è stato istruito in un modo diverso e perciò capisce dei comandi diversi. Questi comandi sono espressi nel linguaggio BASIC. Mano a mano che li imparerete, vi accorgete che parlare col computer è molto facile. Proviamo ad usarne qualcuno di facile, subito.

ALCUNI GIOCHETTI CHE FIDO NON SA FARE ———

Questo con cui cominciamo è facile—anche se Fido non la pensa così.

`WIDTH 10`

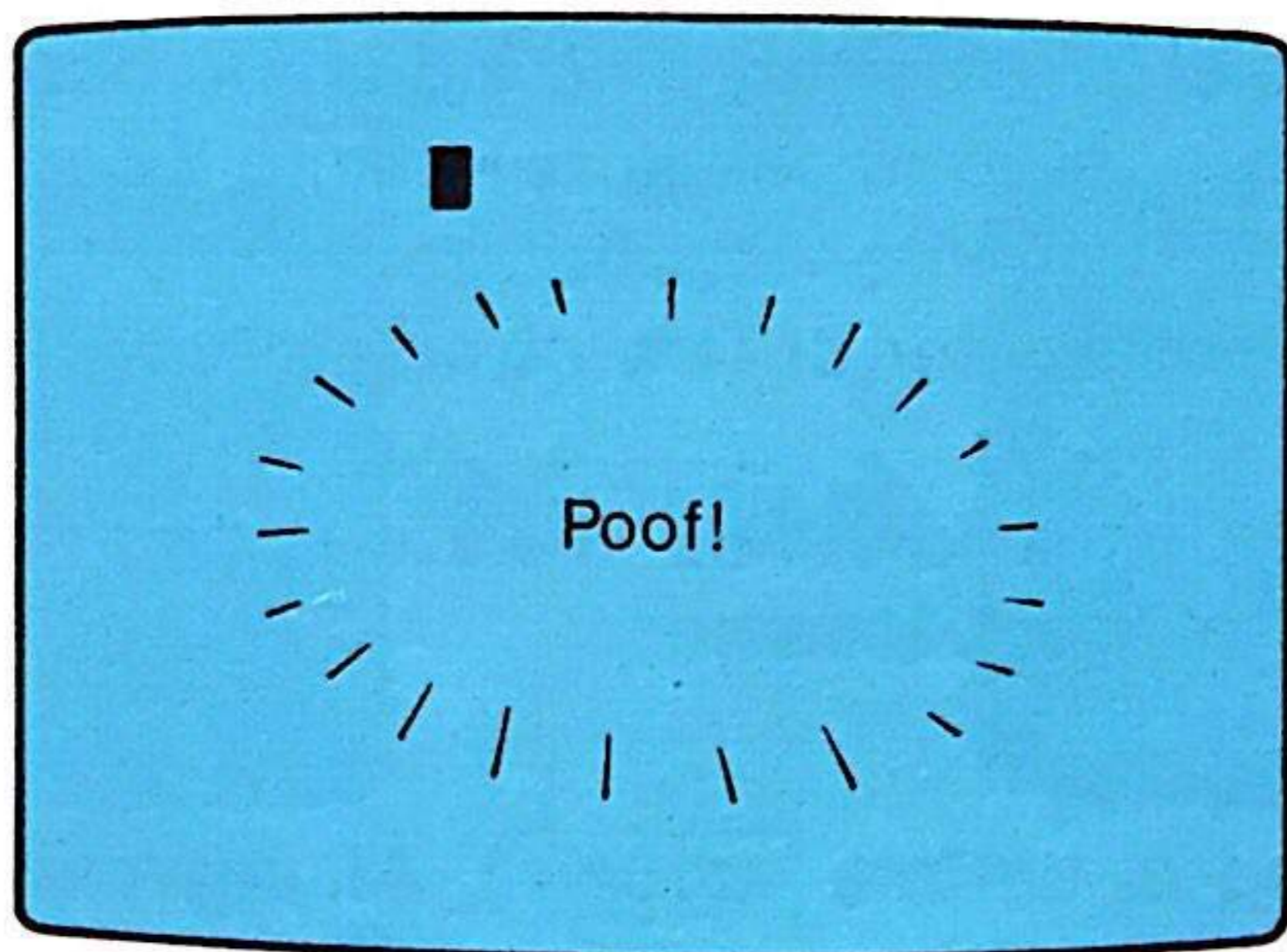
Non è difficile battere questo sulla tastiera; stiamo però attenti a premere una volta la barra di spazio per mettere uno **spazio** tra la parola WIDTH e il numero 10.

`W I D T H 1 0`

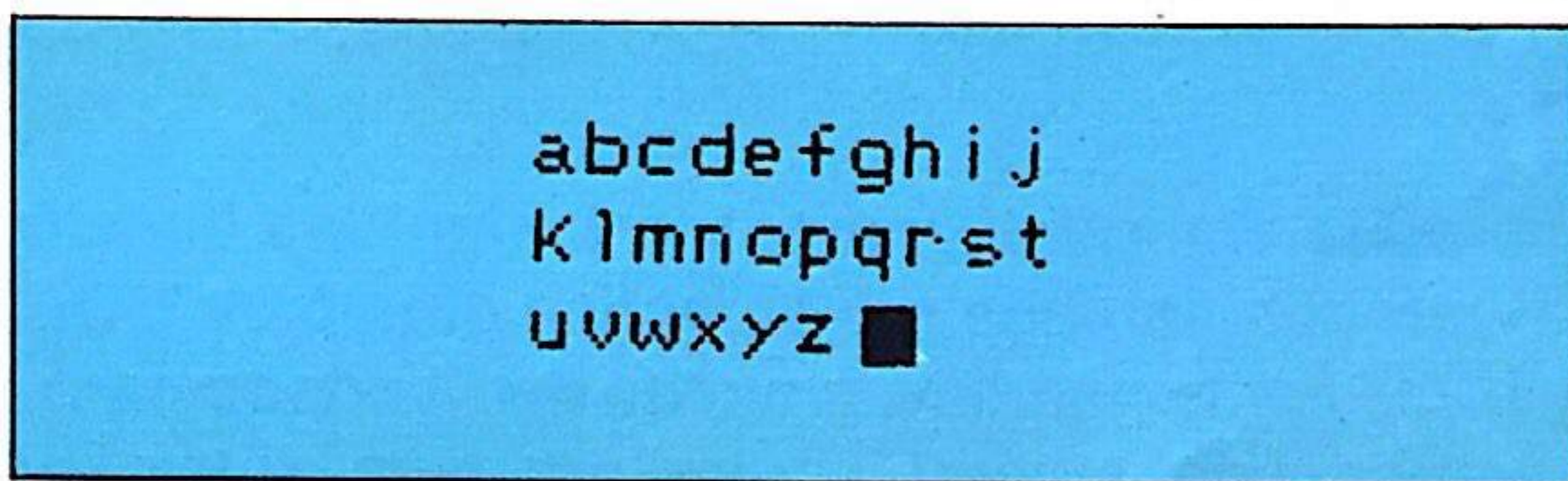
Non c'è bisogno di preoccuparsi di fare errori. Se sbagliamo basta spostare il cursore come spiegato nel manuale delle istruzioni e correggere gli errori. Il Computer Sony dimentica tutti gli errori non appena li correggiamo. Quando sullo schermo del nostro computer è visualizzato questo messaggio,

`WIDTH 10 ■`

siamo pronti a dare il comando. Premiamo il tasto `RETURN` e che cosa succede?



La scritta è scomparsa e il cursore si trova quasi in mezzo allo schermo. Che cosa significava il comando `WIDTH 10`? Battiamo qualcosa sulla tastiera e lo scopriremo subito. Torniamo a qualche **immissione**: battiamo l'alfabeto, senza spazi e senza `RETURN`.

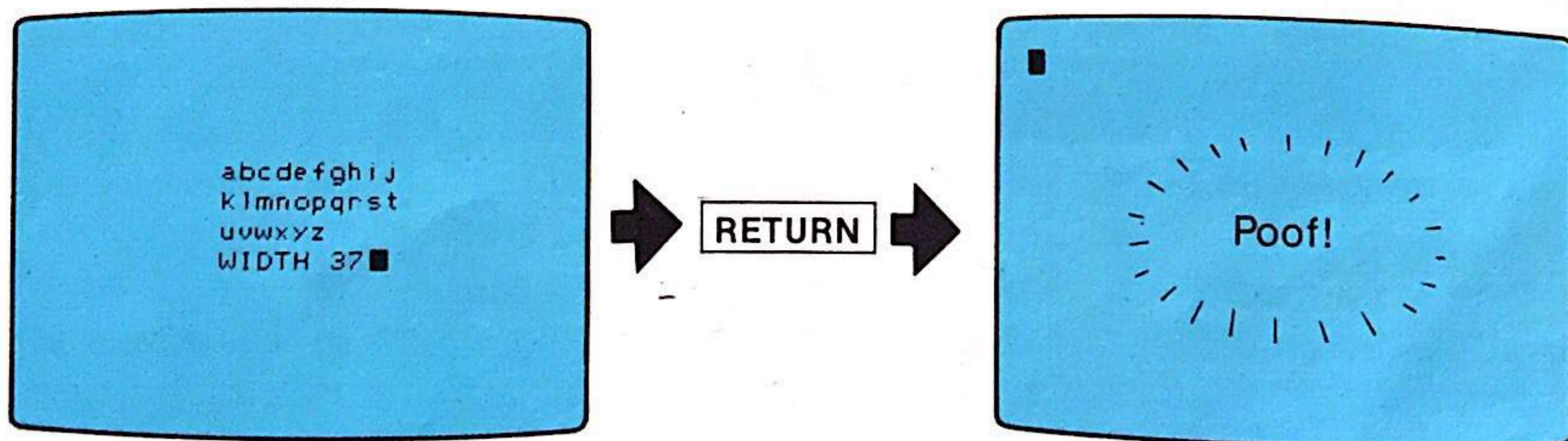


Quante lettere ci sono in ogni riga? Dieci. Senza bisogno di ulteriori comandi, il nostro computer ha capito che vogliamo righe composte di soli **dieci caratteri**. `WIDTH 10` è uno dei comandi che il computer capisce e quando lo sente fa apparire sul suo schermo righe composte di dieci caratteri ciascuna.

Proviamone un altro—ma prima, premiamo insieme i tasti `CTRL` e `STOP` per cancellare le ultime immissioni e preparare il computer per il comando **successivo**:

```
WIDTH 37
```

Premiamo ora il tasto `RETURN`.



Cos'è successo stavolta? Lo schermo è di nuovo vuoto e il cursore si trova in alto a sinistra. Abbiamo detto al computer di scrivere righe di 37 caratteri ciascuna e tutto quello che batteremo d'ora in poi, sarà visualizzato in righe di 37 caratteri, fino a quando non cambieremo il comando. (A proposito, ogni volta che accendiamo il computer, sullo schermo verranno automaticamente visualizzate righe di 37 caratteri, a meno che non diciamo al computer di fare diversamente).

Proviamo ora con un altro comando che Fido non può capire. Premiamo prima **CTRL** e **STOP** e poi battiamo:

COLOR 8

(Non dimentichiamo di battere **RETURN**). All'improvviso tutte le lettere sullo schermo sono diventate rosse! Proviamo a stampare qualche altra parola o altre lettere. Sono tutte dello stesso colore.

Perché non cambiamo colore? Prima di tutto, portiamo il cursore sulla sinistra dello schermo premendo **CTRL** e **STOP**. Poi, battiamo **COLOR 15** e quindi il tasto **RETURN**—quello che battiamo ora appare di nuovo in bianco.

Non è difficile capire che per il computer 15 significa bianco e che 8 significa rosso—ma non sarebbe più semplice stampare **COLOR ROSSO** o **COLOR BIANCO**? Per le persone, e anche per il nostro caro Fido, le parole sono più semplici dei numeri. I computer, però, si trovano più a loro agio con i numeri che con qualsiasi altra cosa: perciò usiamo i numeri per indicare le cose che vogliamo far capire al computer. Il BASIC è un linguaggio che spesso usa i numeri al posto delle parole.

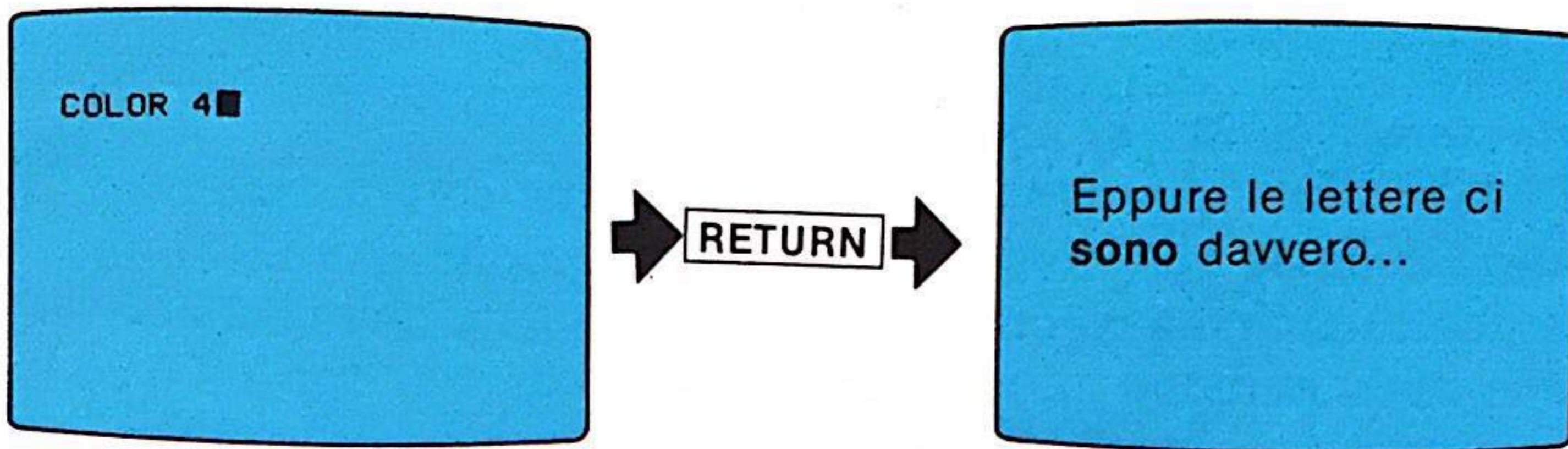
Quando vi sarete abituati a dare comandi al vostro computer, userete i numeri per indicare nomi, luoghi, colori o frasi che volete far ricordare al computer. Non è molto diverso dall'usare i numeri per indicare l'automobile o la casa di persone diverse—e per il computer è il modo di parlare più naturale. Presto saprete usare anche voi senza problemi i numeri, come le lettere, per comporre da soli i vostri programmi BASIC.

Ora sappiamo che 8 significa rosso e che 15 significa bianco. Quanti colori ci sono?

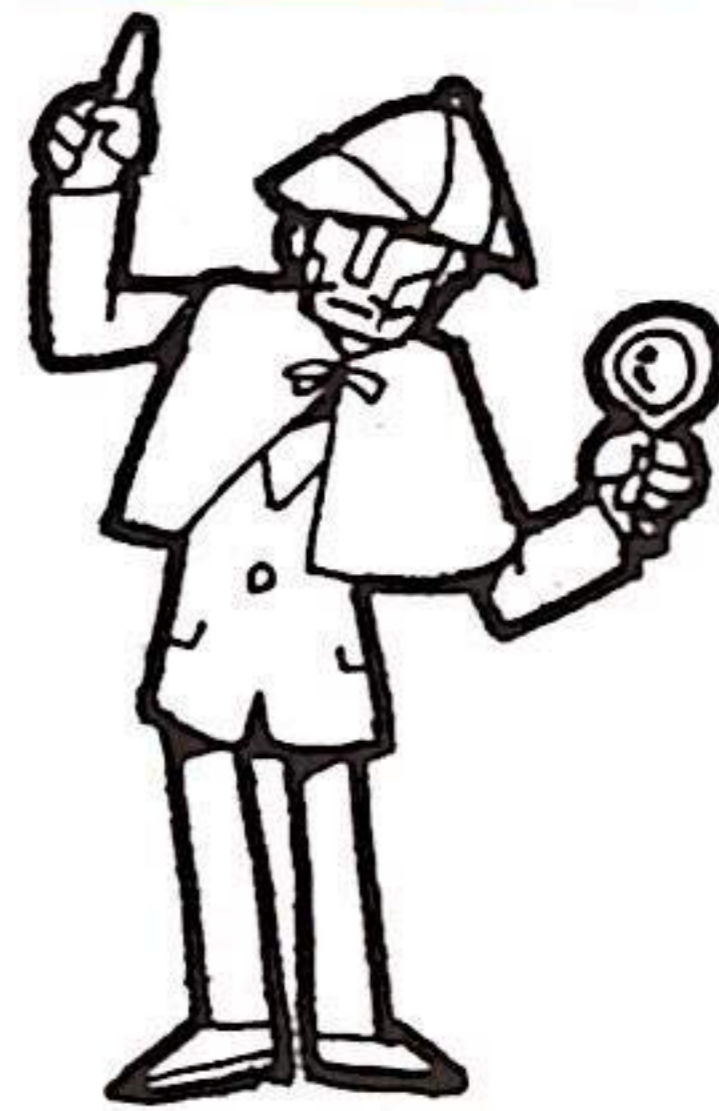
codice	colore	codice	colore	codice	colore	codice	colore
0	trasparente	4	blu scuro	8	rosso	12	verde scuro
1	nero	5	blu chiaro	9	rosso chiaro	13	cremisi
2	verde medio	6	rosso scuro	10	giallo scuro	14	grigio
3	verde chiaro	7	azzurro	11	giallo chiaro	15	bianco

In tutto ci sono sedici colori e può essere bello giocarci. Tanto per divertirci, proviamo questo:

COLOR 4



Lettere blu scuro su fondo blu scuro.
Mmm, è sparito tutto.



Usiamo la tabella dei colori e proviamo sullo schermo i diversi colori. Siamo liberi di cambiare i colori del nostro computer—a seconda dell'umore che abbiamo.

Perché non facciamo qualcosa di un po' più complicato?

COLOR 1000

(Non dimentichiamo `RETURN`). Nella lista dei colori non c'è il numero 1000 e il computer lo sa. Un "piii" immediato ci avverte che il computer ha una risposta pronta per noi. La risposta è "Illegal function call", che nella lingua del computer significa: "Non mi puoi ingannare".

Infatti, è impossibile ingannare il nostro computer Sony nei comandi. Supponiamo di voler battere

COLOR 6

ma di commettere un errore e battere invece

CILOR 6

Se premiamo il tasto `RETURN` senza accorgerci dell'errore, il computer risponderà con un altro "piii". Stavolta il messaggio sarà "Syntax error".

Tutti fanno qualche errore di battitura, naturalmente—perfino i professionisti della Sony! È per questo che sono stati previsti i tasti **BS** (arretramento di spazio), **DEL** (cancellazione), **INS** (inserimento) e quello per il movimento del cursore (vedi il Manuale delle istruzioni). Impareremo molto presto a correggere gli errori.

Quante parole BASIC ci sono?

Ora che abbiamo imparato i primi due comandi BASIC, WIDTH e COLOR potremo verificare quanto sia fedele il nostro Computer Sony nel rispondere a comandi immessi correttamente. Fa quello che gli viene detto se gli viene detto in modo corretto, ci manda messaggi d'errore e ce li spiega se usiamo un comando in modo sbagliato o se ne usiamo uno che non sia nella lista.

Quante parole BASIC ci sono in tutto? Ce ne sono circa 100, che dicono al computer di suonare della musica, di eseguire disegni, di risolvere problemi di matematica e tante altre cose. Il Manuale di riferimento per programmazione MSX-BASIC li spiega tutti. Per cominciare a usare il computer, tuttavia, ce ne servono solo alcuni e quindi non è il caso di preoccuparsi anche se 100 può sembrare all'inizio una quantità enorme.

Anche il nostro Computer Sony ci insegnerà qualcosa fin che stiamo imparando: conosce circa 35 diversi **messaggi di errore** come "Syntax error" e "Illegal function call" che abbiamo appena visto. Ci dirà sempre cosa c'è che non va se gli daremo dei comandi che non capisce.

STAMPIAMO LA RISPOSTA _____

Fido ha i suoi giochi preferiti e dare la zampa è uno di questi. Al nostro Computer Sony piacciono altri giochi. La matematica è una delle cose per cui va matto (e che gli riesce meglio).

Battiamo

PRINT 3+5

Dov'è il segno (+)? È sopra il segno "=" ed appare sullo schermo quando premiamo il tasto **SHIFT** e il tasto "=" **contemporaneamente**.

Premiamo quindi il tasto **RETURN** per introdurre questo comando nel computer.


```
PRINT 3+5
```

```
8
```

```
OK
```



Ce l'abbiamo fatta! Abbiamo usato il computer per risolvere un problema e stampare la risposta. Proviamone un altro:

```
PRINT 100-10
```

Per il segno meno, come vedremo, non c'è bisogno di usare il tasto **SHIFT**.

Usiamo quindi **RETURN** per introdurre il comando.

```
PRINT 3+5
```

```
8
```

```
OK
```

```
PRINT 100-10
```

```
90
```

```
OK
```



La risposta che ci dà il computer è naturalmente 90. Questi sono problemi semplici, ma se chiediamo al nostro computer di addizionare o sottrarre trilioni e bilioni, lo farà in un attimo.

E la moltiplicazione? Non c'è problema. Bisogna solo ricordare che per la moltiplicazione il computer ha un **segno speciale**. Invece di 7×9 il computer capisce $7 * 9$.

Battiamo PRINT $7 * 9$ e premiamo poi **RETURN**.

```
PRINT 7*9
```

```
63
```

```
OK
```



Per la divisione si usa il tasto **/**. Invece di PRINT $120 \div 40$, battiamo

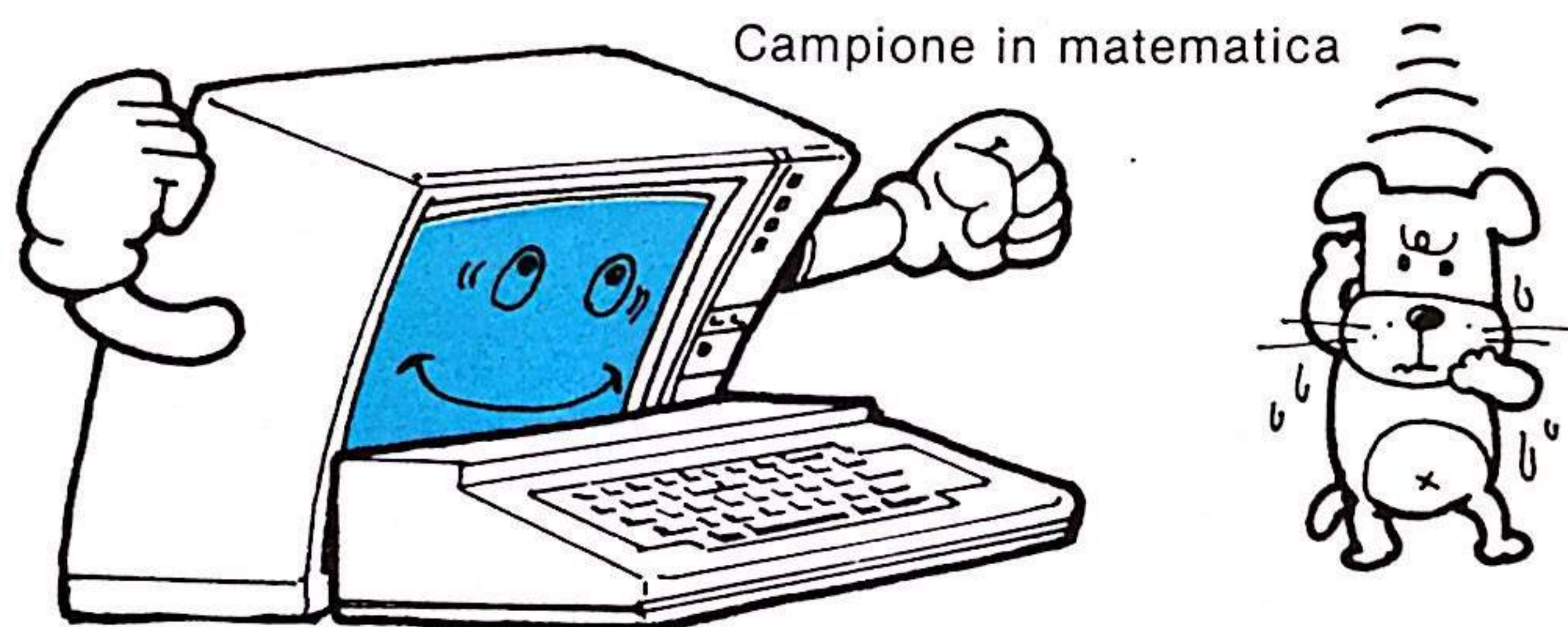
```
PRINT 120/40
```


E quando premiamo il tasto **RETURN**, il computer, naturalmente, ci darà la risposta esatta.

```
FRINT 120/40
  3
OK
■
```

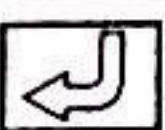
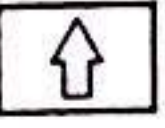
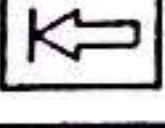

Siete pronti a provare con numeri grandi (7654321×18)?
O problemi difficili $(2 + 9) \times (6 - 8)$?

Coraggio, tutto quello che vi pare. Può capitare che il computer dia delle risposte incomprensibili, ma non è il caso di preoccuparsene. Facciamo un passo per volta, pensiamo ai risultati e in pochi minuti saremo capaci di fare cose incredibili.



Nota

Il vostro computer MSX potrebbe avere parecchi tasti diversi da quelli illustrati in questo manuale.
Fate riferimento alla tabella seguente.

Sulla tastiera	In questo manuale
	RETURN
	SHIFT
	BS
	CAP
SUP	DEL
EFE	HOME

NUMERI, LETTERE, VARIABILI

Come...

- Stampare parole e frasi
- Creare una variabile
(un simbolo con valore mutevole)
- Usare le variabili in matematica
- Dare nomi diversi alle variabili



Abbiamo usato il comando PRINT per far risolvere al computer dei problemi di matematica e finora si è comportato come un qualsiasi calcolatore. È però giunto il momento di vedere quali altri tipi di calcoli è in grado di fare questa macchina. Per cominciare, cerchiamo di scoprire qualcosa di più sul comando PRINT e di fare nello stesso tempo un po' di pratica con la tastiera battendo.

```
PRINT "PAOLO E MARIA"
```

I segni delle virgolette (") si trovano sopra il segno ' e il tasto va quindi premuto insieme con **SHIFT**. Dopo aver immesso tutto quello che ci interessa, premiamo il tasto **RETURN**.

```
PRINT "PAOLO E MARIA"  
PAOLO E MARIA  
OK  
■
```

Ovviamente, il computer "legge" le virgolette e interpreta il comando PRINT in questo modo: deve stampare quello che è contenuto all'interno delle virgolette ma non le virgolette. Facciamo qualche altro esempio:

```
PRINT "ABC TO XYZ"  
ABC TO XYZ
```

```
PRINT "How are you?"  
How are you?
```


Naturalmente, bisogna premere ogni volta il tasto **RETURN**. Non dovrebbe essere difficile ricordarlo ormai, perciò non lo diremo più d'ora in poi: solo, bisogna premere il tasto **RETURN** ogni volta che si vuole introdurre nel computer un comando.



NUMERI O LETTERE: SONO LA STESSA COSA —

Proviamo a fare qualcosa di leggermente diverso.

```
PRINT "3+5"
```

Assomiglia un po' a quello che abbiamo fatto alcune pagine indietro. Allora, abbiamo chiesto al computer di eseguire **PRINT 3+5** e il computer, molto gentilmente, ci ha stampato 8—perché non c'erano le virgolette.

Battendo il tasto delle virgolette otteniamo invece:

```
PRINT "3+5"  
3+5  
OK  
■
```

In questo caso, il comando **PRINT**—usato insieme **con le virgolette**—dice al computer di leggere quello che è contenuto tra le virgolette e di farlo apparire così com'è sullo schermo. A causa delle virgolette, il computer legge "3+5" come tre simboli da stampare—non come un problema di matematica da risolvere.

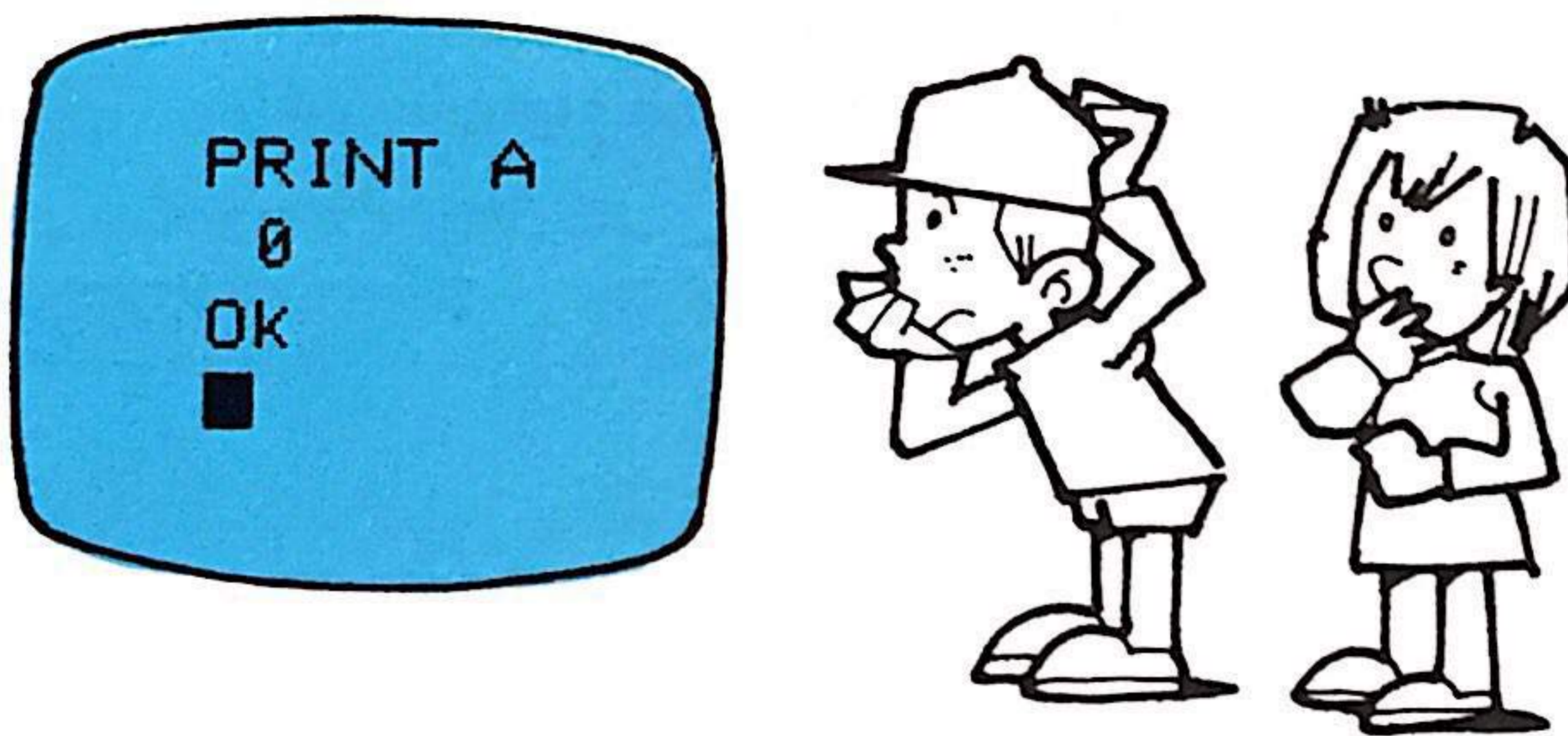


Tutti e due vengono fatti nello stesso modo

Questo è quello che succede quando un numero viene posto tra virgolette.

UNA LETTERA DIVENTA UNA VARIABILE _____

Ora, impareremo un'altra cosa molto interessante sul comando PRINT. Cosa succede quando una lettera senza virgolette viene posta dopo PRINT?

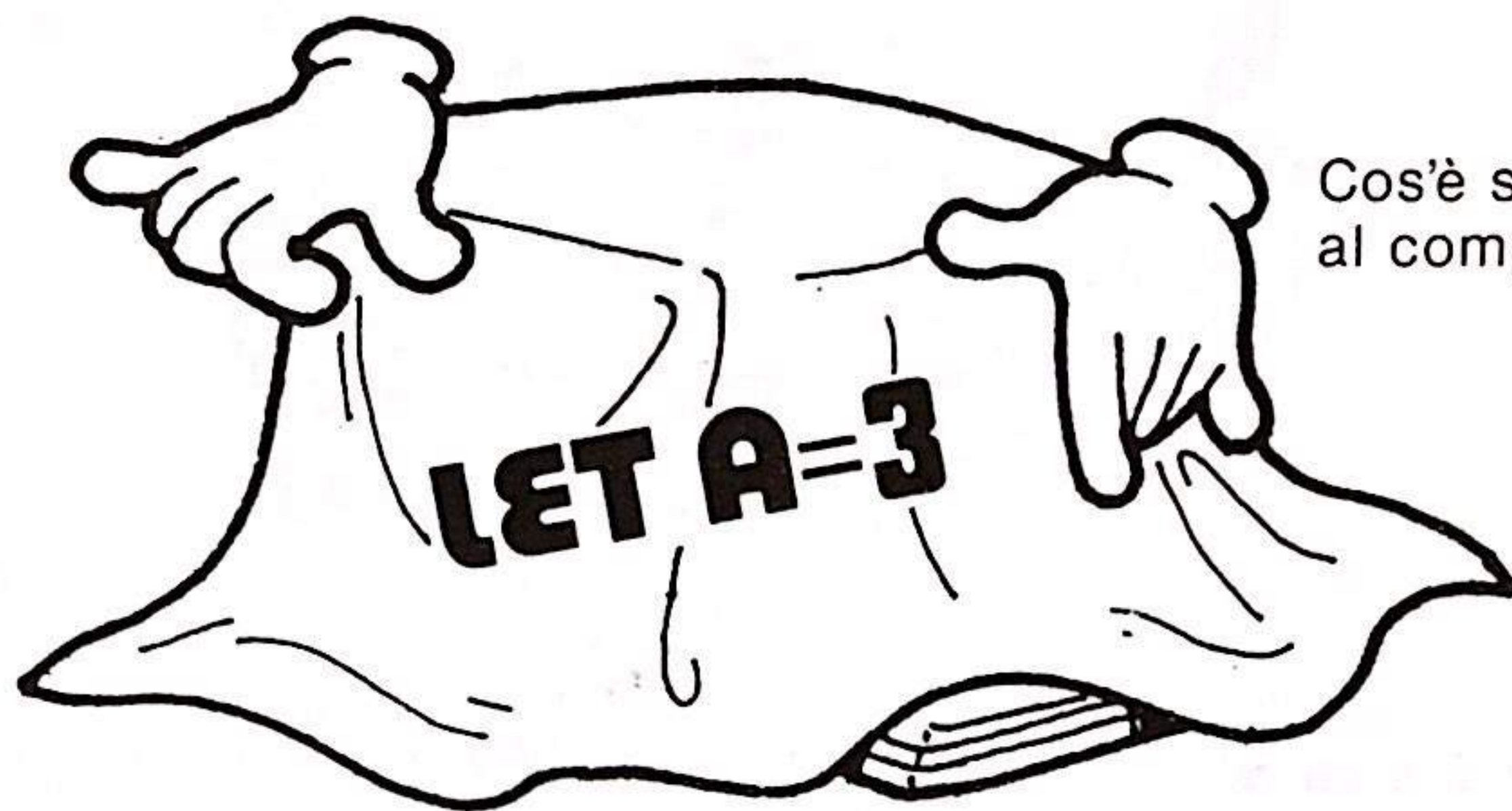


Cosa succede qui? Niente "piii"! Niente "Syntax error"! C'è solo uno 0 e l'Ok che ci dice che il computer ha eseguito il nostro comando e sta aspettando quello successivo. Ma che comando gli **abbiamo dato**?

Se non c'è nessun messaggio d'errore significa che PRINT A era un comando perfettamente accettabile. Che cosa ordina al computer questo comando? Per scoprirlo facciamo:

```
LET A=3
```


Quando premiamo il tasto `RETURN` non c'è nessun messaggio d'errore, ma non succede nemmeno poi molto, a parte quel rassicurante `Ok`. Il fatto che il computer abbia accettato il nostro comando significa però che qualcosa è successo dentro la macchina.



Cos'è successo dentro al computer?

Ora, mettiamo insieme questi due comandi:

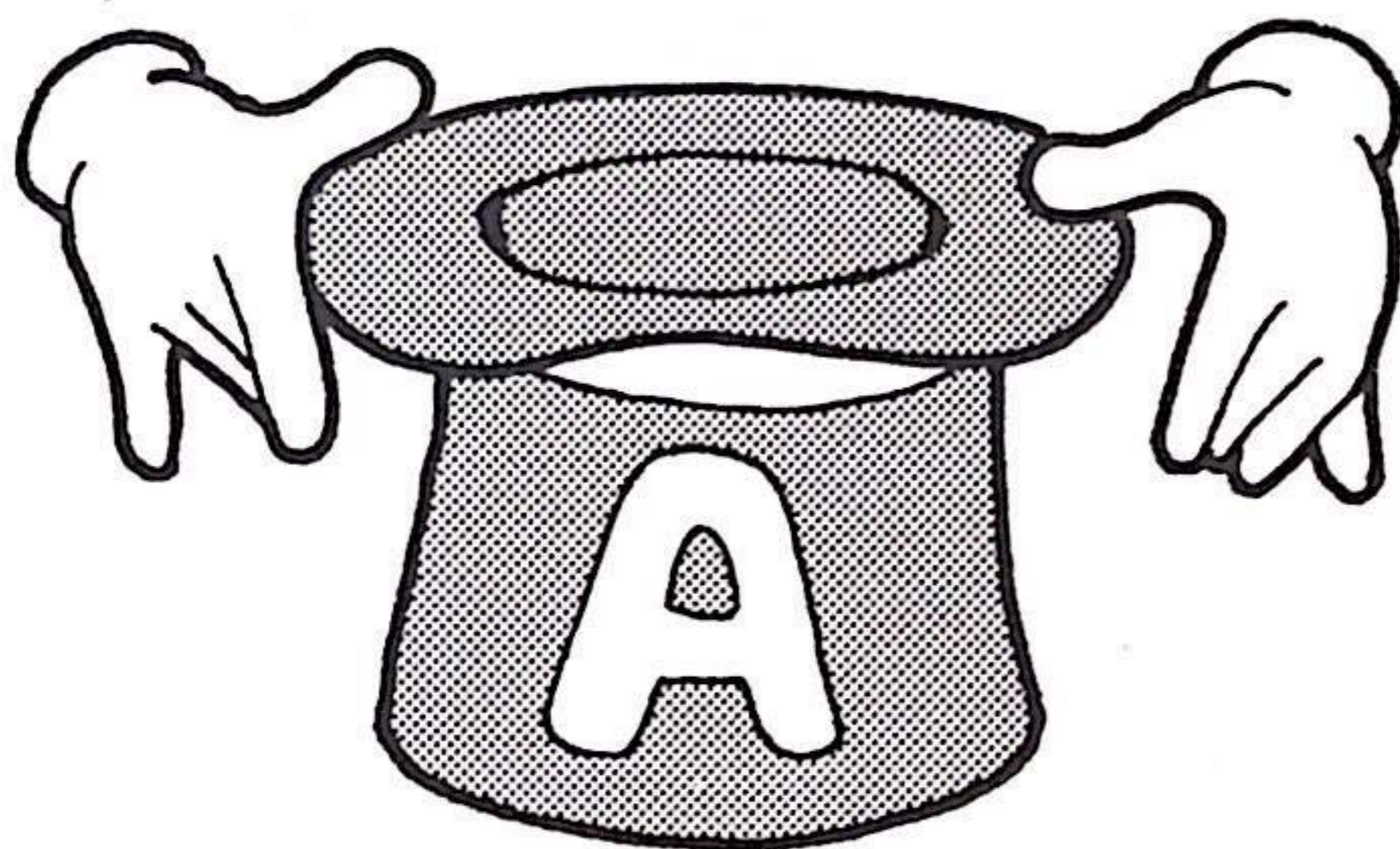
```
LET A=3
OK
PRINT A
  3
OK
■
```

Cominciamo a farci un'idea di quello che sta succedendo ma proviamo un'altra volta:

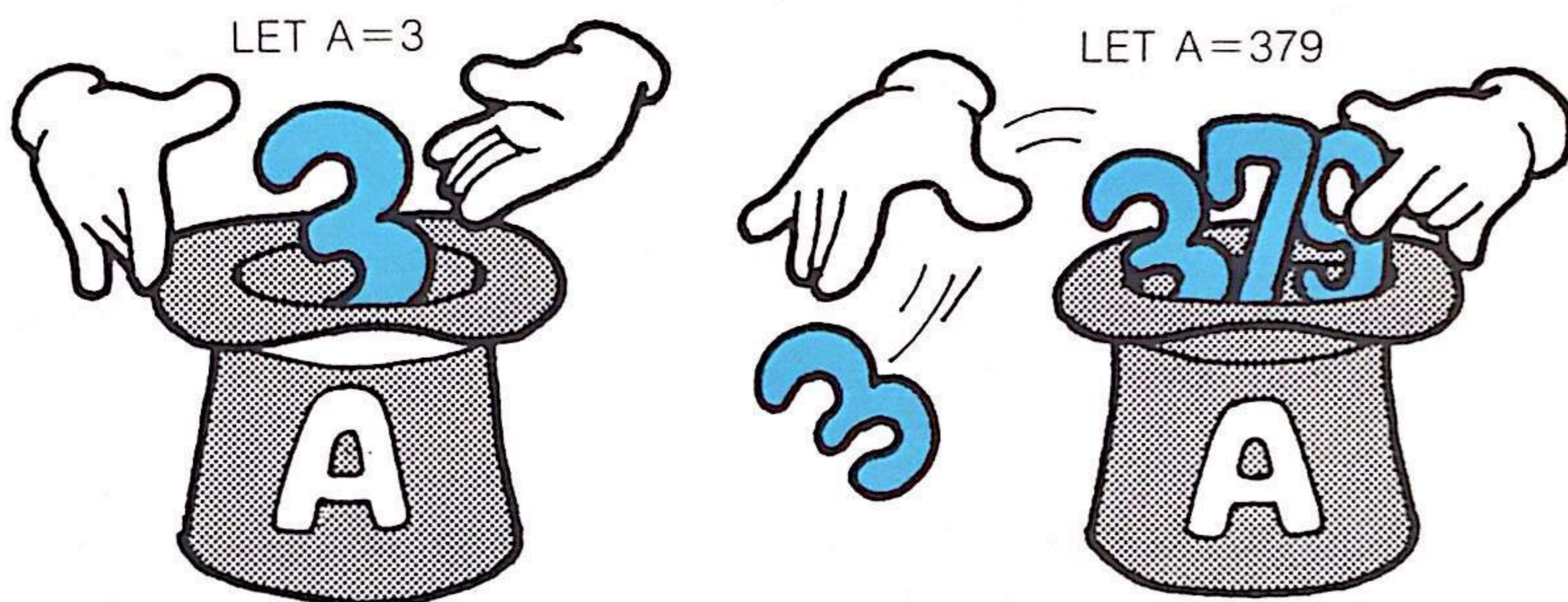
```
LET A=379
OK
PRINT A
 379
OK
■
```

Un attimo fa **A** era **3** ma ora è **379**. Se vogliamo possiamo farla variare ancora scrivendo semplicemente `LET A=` e immettendo un altro numero. Per questo la chiamiamo **variabile**.

Questa **A** dunque non è proprio una lettera dell'alfabeto. Piuttosto, quando appare dopo il comando `PRINT` senza virgolette, è qualcosa come un contenitore che può tenere qualsiasi numero noi le assegniamo (`LET`).



Le variabili ricordano un po' il cappello di un mago, nel senso che, come nel cappello di un mago, vi si possono mettere dei valori che si possono tirare fuori quando se ne ha bisogno.



Se battiamo `LET A=3`, mettiamo 3 nel cappello e se vogliamo tirarlo fuori basta che premiamo `PRINT A`. Con un solo, semplice comando (`LET A=379`), possiamo cambiare il valore contenuto dentro il cappello da 3 a 379 e farlo venire fuori quando ne abbiamo bisogno.

`LET` è quindi un comando che assegna un valore specifico alla nostra variabile. Le variabili sono usate in molti modi diversi nel BASIC e verranno usate molto spesso in questo libro. Per comodità, possiamo scrivere questo comando più velocemente, eliminando la parola `LET` in questo modo:

```
A=3
```

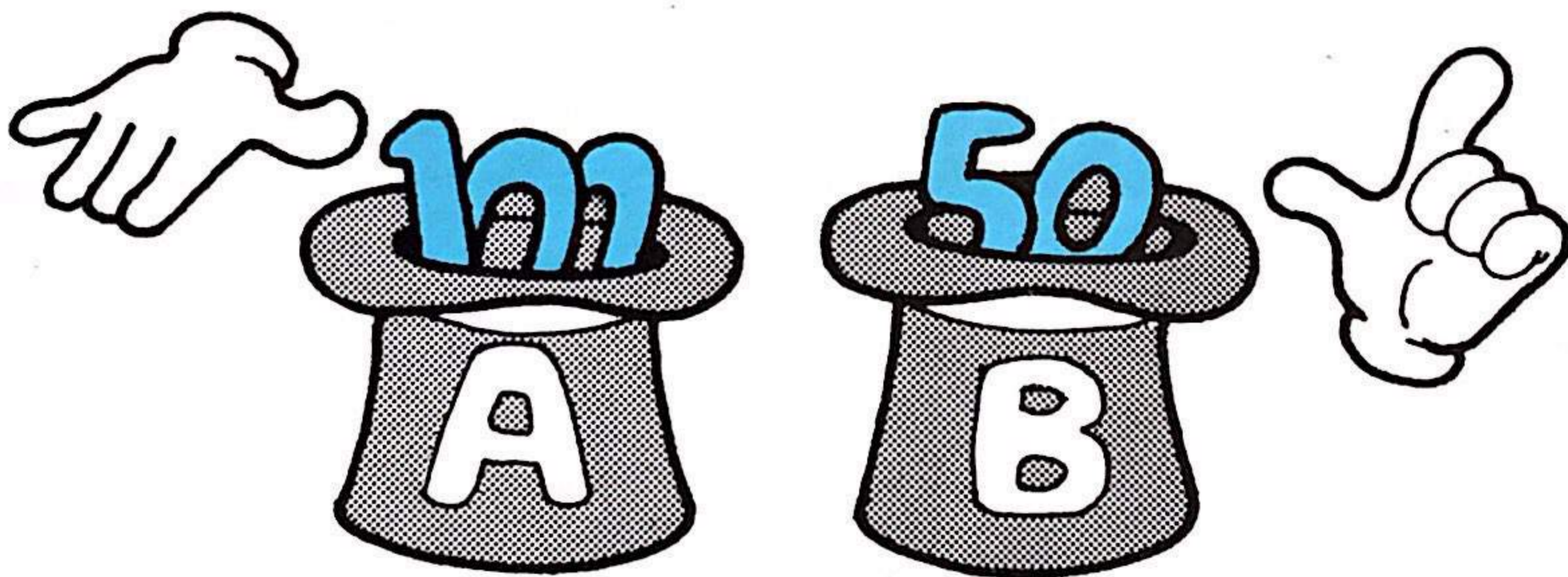
ha lo stesso significato di `LET A=3`.

Ora, per essere sicuri di non dimenticare niente, facciamo un piccolo riassunto:

A=100	Diamo ad A il valore 100
OK	
PRINT A	Facciamo apparire A
100	Eccola
OK	
PRINT "A"	Facciamo apparire la lettera "A"
A	(non la variabile A)
OK	
B=50	Diamo un valore a B
OK	
PRINT B	Facciamo apparire B
50	Eccola
OK	
■	

Poco fa abbiamo dato al computer il comando PRINT A e lui ci ha risposto con 0. Ciò significa che ogni variabile ha il valore zero fino a quando non le diamo un valore diverso da zero.

Una variabile può essere rappresentata sullo schermo con una lettera qualsiasi. Abbiamo già usato A e B abbiamo dato loro valori diversi.



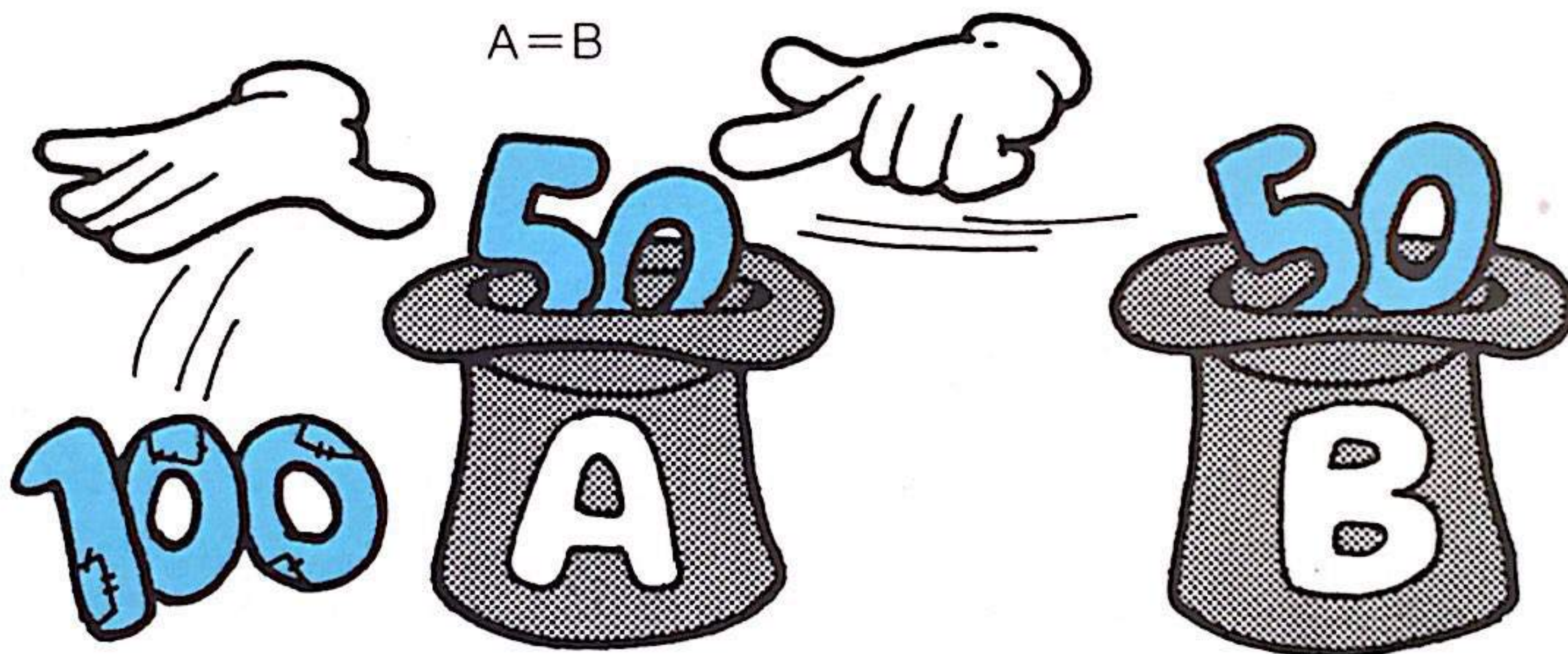
Proviamo ora a vedere in che modo possiamo impiegare queste variabili.

Abbiamo già detto al computer che $A=100$ e $B=50$. Diamo ora tre comandi: $A=B$, PRINT A e PRINT B. Non è difficile indovinare il risultato, ma proviamo ugualmente, per fare un po' di pratica.


```
A=B
OK
PRINT A
  50
OK
PRINT B
  50
OK
■
```

Guardiamo bene la prima riga. Quando abbiamo battuto $A=B$, abbiamo detto al computer di dare ad A il valore di B. Precedentemente, abbiamo cambiato il valore di A da 3 a 379 e il vecchio valore è semplicemente scomparso.

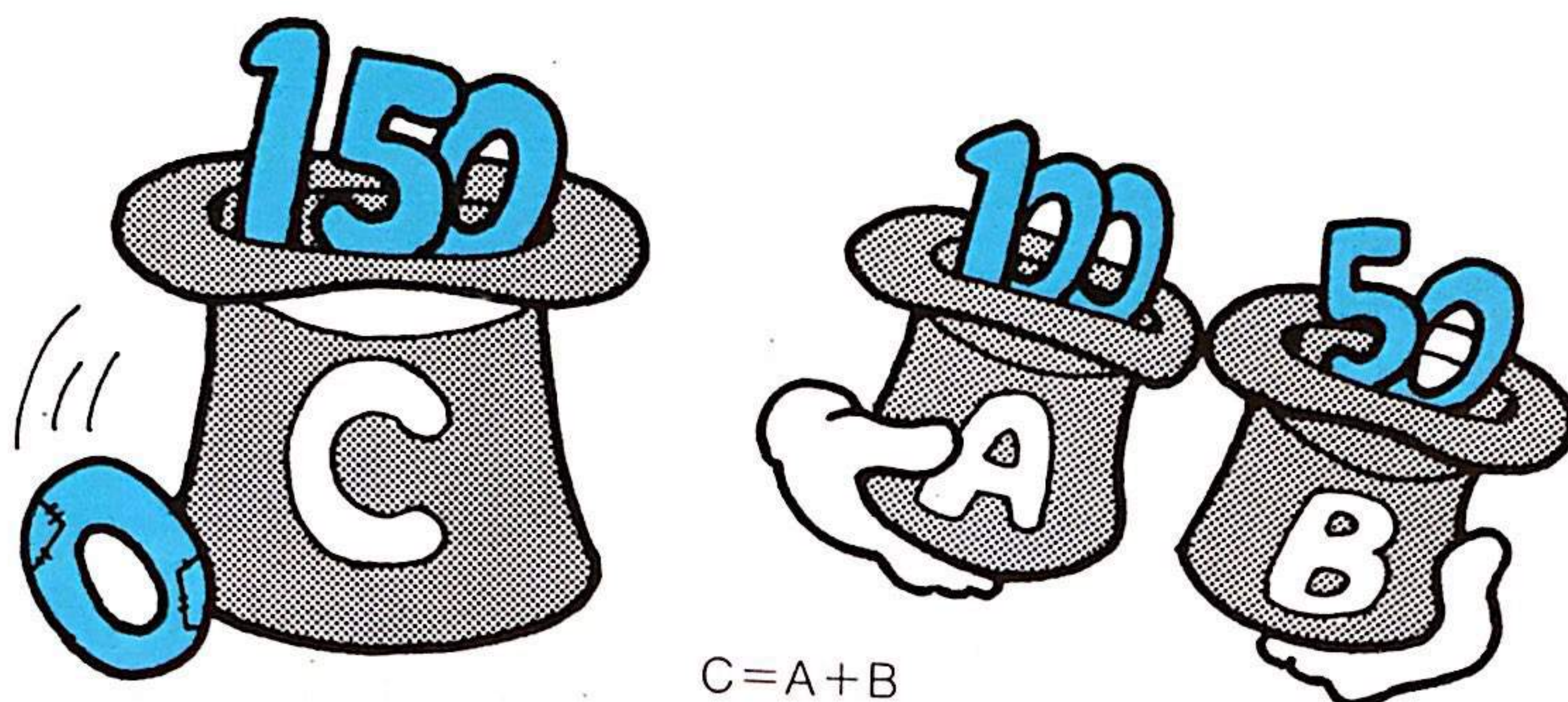
Anche qui succede la stessa cosa, con A che cambia da 100 a B o a 50.



Ora possiamo provare a fare un po' di matematica. Battiamo questi quattro comandi:

```
A=100
OK
B=50
OK
C=A+B
OK
PRINT C
  150
OK
■
```

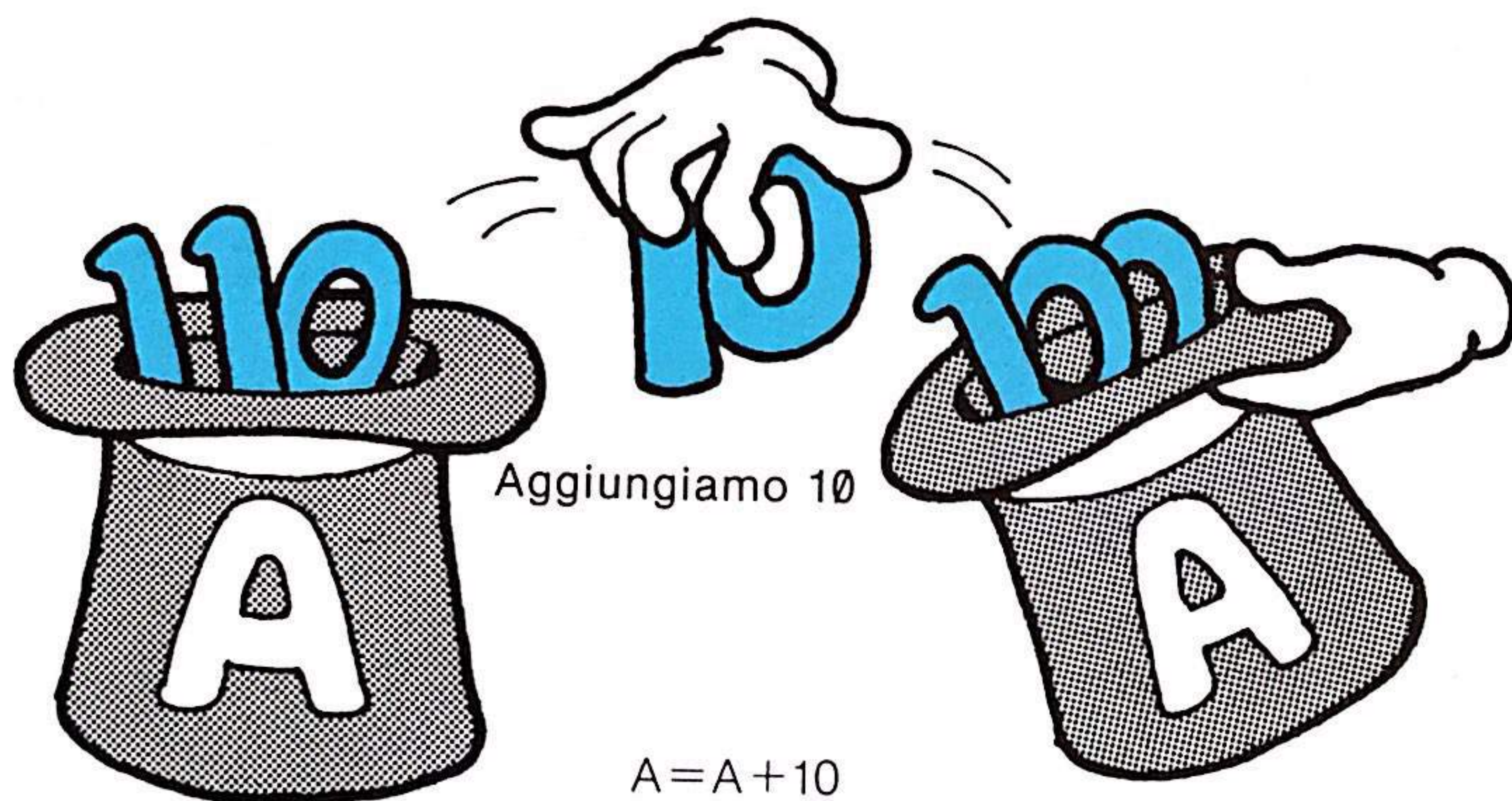

Qui abbiamo tre contenitori, A, B e C e il computer esegue dei calcoli per scoprire quanto mettere nel contenitore C.



Ora eccone un altro ancora più interessante:

```
A=A+10
OK
PRINT A
  110
OK
■
```

È un po' complicato? Ad A era stato "assegnato un valore" di 100. Tuttavia, leggendo questo nuovo comando il computer ha compreso il nostro comando di "assegnare ad A un valore di 10 unità più grande di quanto fosse prima". Perciò A è ora 110.



Possiamo provare di nuovo ad usare lo stesso comando se vogliamo. Prima di farlo però, dovremmo essere in grado ogni volta di prevederne il risultato. Il nostro computer **ricorderà sempre** il valore assegnato alla variabile.

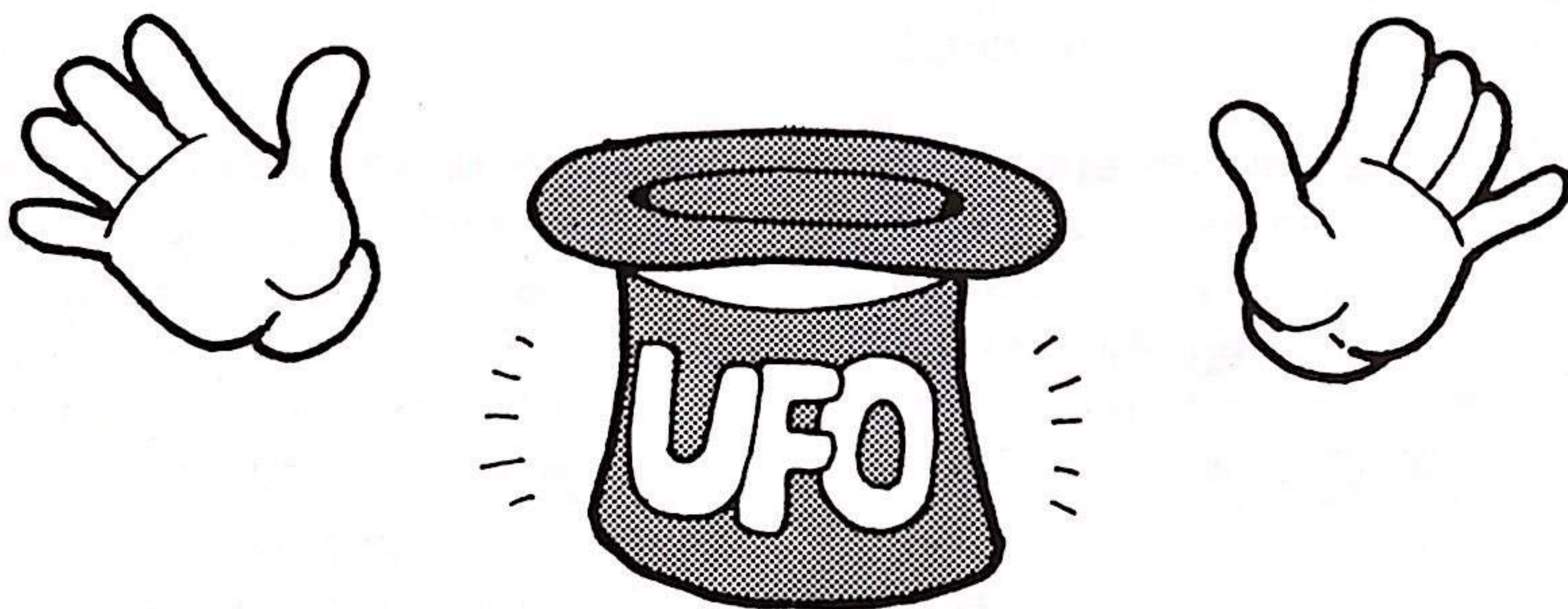
Qualcosa di più sulle variabili

Ora sappiamo cosa sono le variabili e come funzionano. Fra poco vedremo come vengono usate e proveremo ad usarle nei programmi e nei giochi. Le variabili possono essere usate in moltissimi modi. Una variabile può essere usata per ricordare il punteggio di una gara (un numero che varia) ogni volta che vengono aggiunti o sottratti nuovi punti. Altre variabili possono indicare le diverse posizioni di una nave spaziale o la velocità di una macchina da corsa.

Abbiamo già usato tre lettere—A, B e C—come variabili. Oltre a queste, possiamo usare quasi **ogni** lettera o gruppo di lettere.

```
UFO=5
OK
PRINT UFO
5
OK
■
```

Come si può vedere, un gruppo di lettere senza spazi tra di loro, in questo caso la parola UFO, può rappresentare un numero o **una cifra**.



Inoltre, è importante ricordare anche che ogni lettera o gruppo diverso, quali A, B o AB, hanno diversi significati.


```
A=3
OK
B=5
OK
PRINT A
  3
OK
PRINT B
  5
OK
PRINT AB
  0
OK
■
```

Qui, abbiamo assegnato ad A il valore di 3 e a B il valore di 5—ma siccome non abbiamo assegnato nessun valore alla variabile AB, il computer fa apparire AB con un valore di 0 (non 35!). Una variabile ha sempre il valore di 0 fino a che non le diamo un valore diverso.

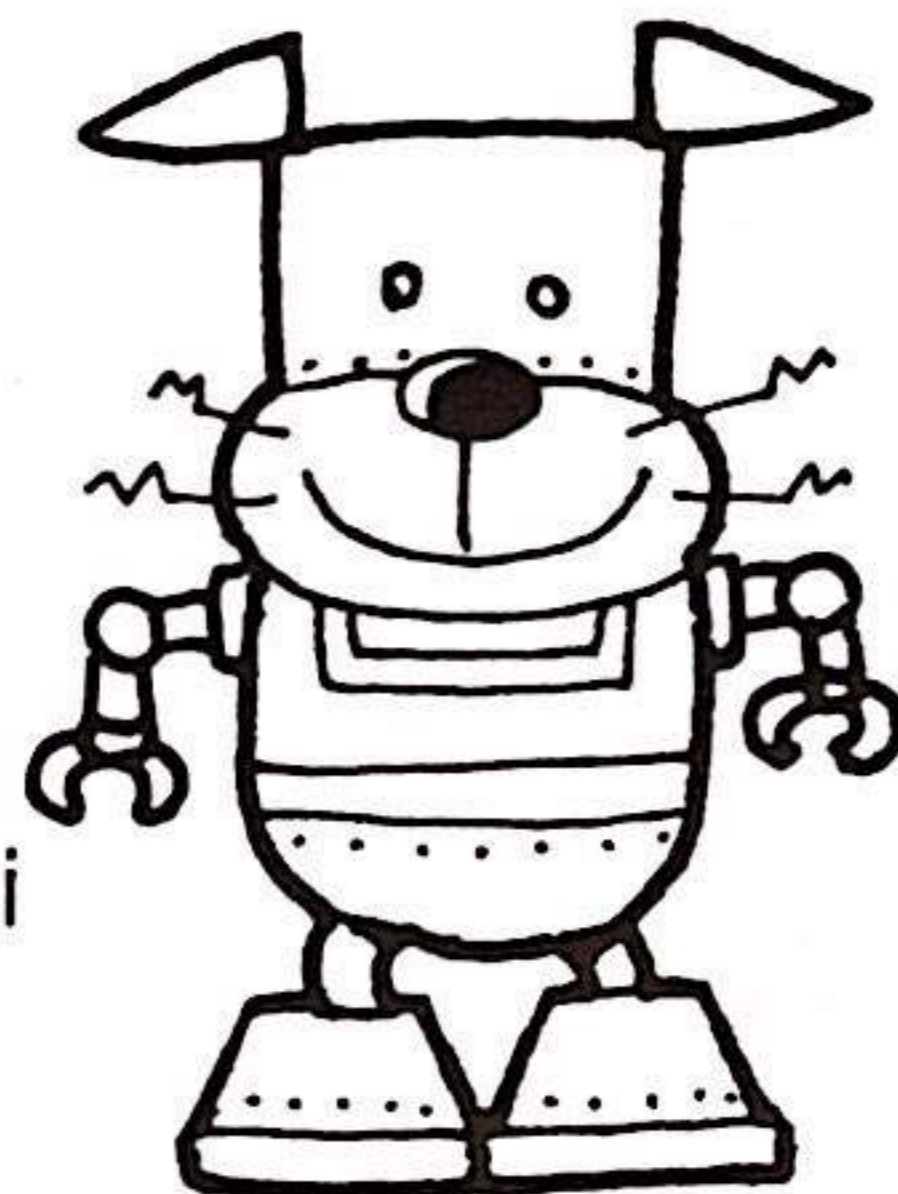
Il nome che assegnamo ad una variabile può essere costituito da due, tre, dieci—o anche 100—**caratteri**, e questi caratteri possono essere sia **numeri** che **lettere**. Tuttavia, **il computer leggerà solo i primi due caratteri** e questi due caratteri devono essere sempre costituiti da lettere. Questo significa che POA, POB, PO1 e PO2 saranno trattati dal computer come se avessero un unico, stesso valore. Bisogna perciò stare un po' attenti a scegliere il nome delle variabili.

Qualsiasi parola che appartenga al linguaggio BASIC **non può** essere usata come nome di una variabile. Se usiamo una parola BASIC con il nostro computer, lui la interpreterà come un **comando** e non come una variabile. Questo significa che LET, GOTO e PRINT, per esempio, non possono diventare nomi di variabili, così come non lo potranno diventare GOTOA, BLET e tutti gli altri comandi che presto impareremo.

FABBRICHIAMO IL NOSTRO PRIMO PROGRAMMA

Come...

- Progettare un programma
- Scrivere un programma campione
- Far memorizzare il programma al computer
- Correggere un programma
- Eseguire un programma
- Leggere sullo schermo i programmi completati
- Cancellare un programma dalla memoria del nostro Computer Sony



Finora ci siamo esercitati sugli elementi fondamentali dell'uso di un computer: far apparire lettere e numeri sullo schermo, usare alcuni dei simboli che il computer capisce perché parte della sua "lingua" e dare semplici istruzioni (comandi). In altre parole, abbiamo detto al computer di **fare** delle cose sullo schermo nel modo in cui noi vogliamo siano fatte. In tal modo, abbiamo già imparato come comunicano la gente e i computer.

Tutto questo, però, non è molto diverso da quello che anche un buon calcolatore tascabile sarebbe in grado di fare. Anche un calcolatore sa addizionare e sottrarre ed eseguire diverse operazioni aritmetiche, mostrandoci sul suo quadrante elettronico tutti i passi del procedimento, fino al risultato.

Ciò che fa del **computer** una cosa entusiasmante per noi (e uno strumento prezioso per gli scienziati e molte e altre persone) è che può fare molto più di questo. Il nostro Computer Sony può:

- immagazzinare nella sua **memoria** una lunga serie di **istruzioni speciali**;
- usare queste istruzioni per eseguire diversi compiti, giochi e trucchi, ogni volta che lo desideriamo;
- capire il risultato di ogni funzione e portarlo alla funzione successiva;
- combinare i risultati di tutti i passi precedentemente eseguiti continuando ad eseguire quelli successivi.

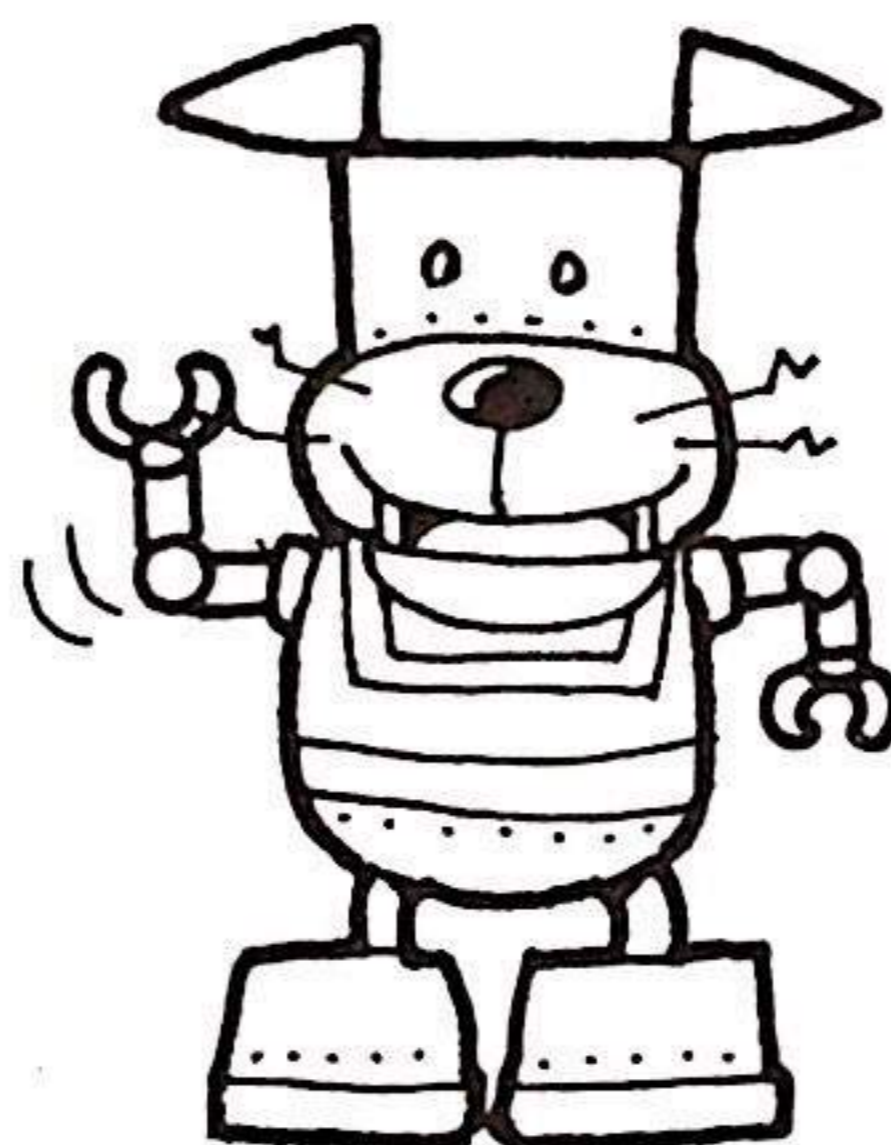
In altre parole, il nostro computer è in grado di seguire automaticamente un **programma** di funzioni completo, molto più velocemente di qualsiasi calcolatore.

La **programmazione** è il processo di introdurre istruzioni in un computer, di dirgli quali funzioni eseguire e a quali ordini ubbidire. È quello che ci permette di usare un computer per creare dei videogiochi personali, di eseguire dei video-disegni, di comporre della musica elettronica. Questo è quanto stiamo per imparare in questo capitolo: stiamo per fabbricare il nostro primo, semplice programma.

Fare un programma per computer non è una cosa difficile. Sappiamo già come dare dei comandi al nostro computer, e un programma non è altro che una serie di comandi. La cosa importante di un programma è l'**ordine di esecuzione**: il **numero** di comandi e l'**ordine** in cui essi vengono dati.

Cercheremo di spiegare un po' più chiaramente i programmi ricorrendo ancora all'esempio di un cane in azione. Siccome Fido sta riposando in questo momento, faremo qualche giochetto con—**Supercane!**

Supercane è un tipo diverso di cane da compagnia: è un robot! (E perché no? Quasi tutto è automatizzato oggi, no?).



Io sono Supercane e so seguire i programmi!

Come abbiamo visto, Fido sapeva fare tante cose, ma, a dire il vero, non erano così difficili. Quando gli gettiamo una palla, Fido la trova, la afferra tra i denti, ce la riporta e la lascia andare. Se sente il comando “qua la zampa”, alza una zampa e ce la lascia prendere. Vogliamo bene a Fido perché è un cane intelligente e per tante altre ragioni—ma lui può eseguire solo sequenze di tre o quattro comandi per volta. Per tenerlo occupato, dobbiamo passare a un altro gioco—un nuovo “programma”.

PROGETTIAMO UN PROGRAMMA

Supercane è diverso. È fornito di un microprocessore in grado di ricordare le cose e prendere delle decisioni, di modo che può eseguire molte più azioni di quelle che potrebbe eseguire Fido in un solo compito. Proprio come un calcolatore, che può risolvere molti più problemi di matematica di quanti ne possiamo risolvere noi nello stesso tempo. Per esempio, possiamo dire a Supercane:

1. Trova la palla e riportacela!
2. Se la palla è bianca, girati tre volte e abbaia!
3. Se la palla è nera, stai sdraiato per 10 secondi!

Il primo comando è facile per Fido. Gli altri due, però, sono molto più complicati: il cane deve essere in grado di identificare la palla come bianca o nera, quindi, deve usare quell'informazione per decidere quale azione compiere e, alla fine, deve compiere correttamente quell'azione.

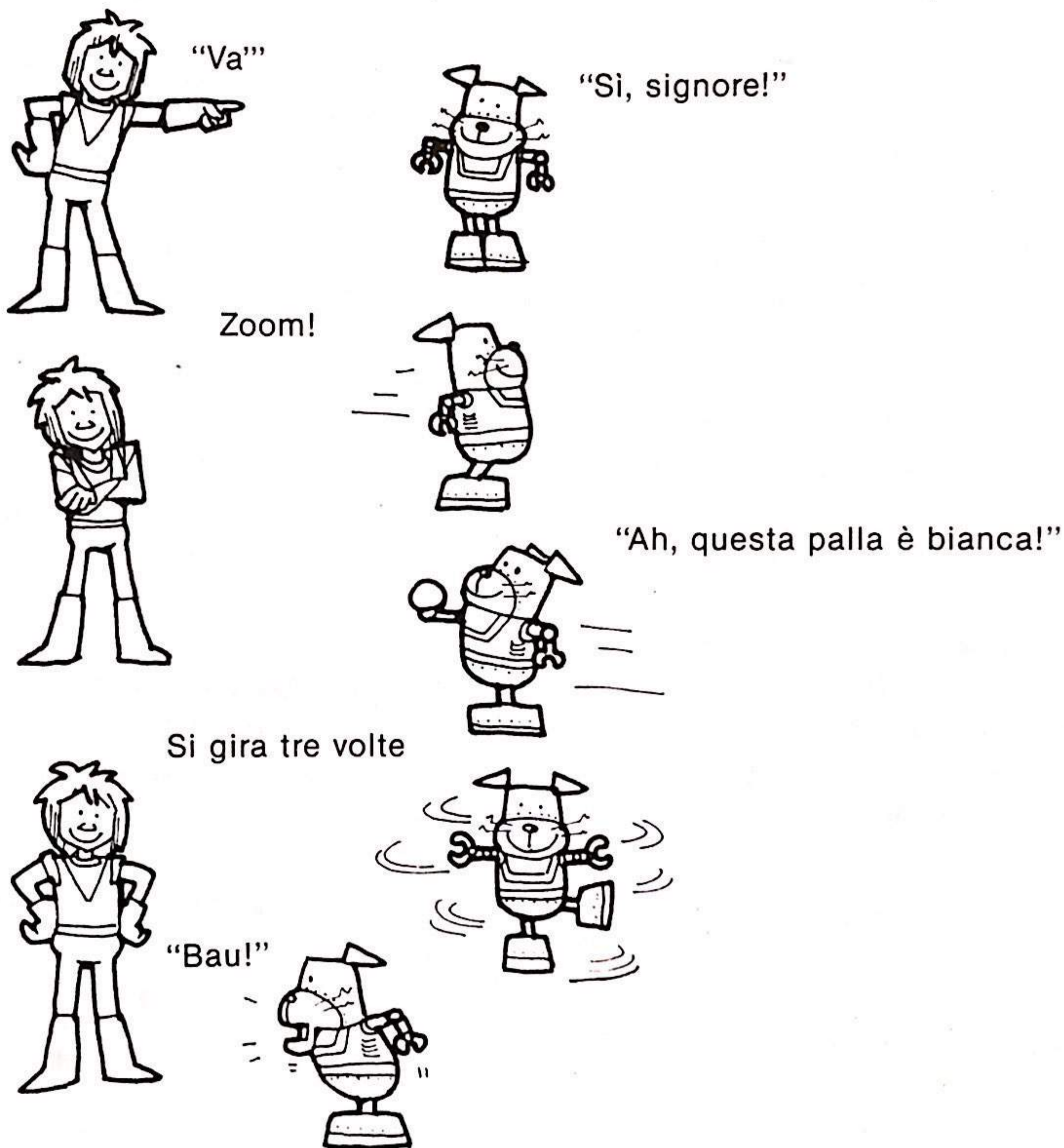
In ogni azione, inoltre, è necessario contare attentamente: il cane deve sapere esattamente quante volte girarsi e quanti secondi stare sdraiato. E per eseguire tutto perfettamente, deve sapere sempre esattamente, in qualsiasi momento, quante volte si è girato o quanti secondi sono passati e quanti ne rimangono prima di abbaiare e riportare indietro la palla. Supercane deve giudicare, prendere delle decisioni e fare dei calcoli per tutto il tempo in cui sta eseguendo il compito che gli è stato affidato.

C'è un'altra differenza tra Fido e Supercane. Il nostro cane di famiglia è un animale, con un cervello un poco simile a quello di noi esseri umani, di modo che può fare delle cose da solo, senza che glielo ordiniamo. Se Fido vede un gatto, abbaia; se vede un incendio, scappa. Supercane è invece una macchina. Sa fare cose estremamente complicate, ma non è capace di "pensare" a queste cose, né di ricordare cosa deve fare. Supercane fa **solo** quello per cui è stato **programmato** e non farà mai niente fino a che un essere umano non gli ordini di mettere in funzione le sue parti—motori, sensori, componenti e il microprocessore—in un determinato ordine.

Perciò, quando scriveremo il programma per i compiti di Supercane, dovremo prima pensare a tutto quello che vogliamo far fare alla macchina—**ogni** minimo passettino, e in quale ordine lo vogliamo eseguito. Qualcosa del genere:

1. Ricevi e identifica il comando "Trova la palla e riportamela!"
2. Usa i sensori dell'immagine per seguire il movimento della palla e localizzarla.
3. Aziona le zampe meccaniche per muovere la palla.
4. Usa i sensori dell'immagine per identificare la palla come "nera" o "bianca".
5. Aziona le parti del corpo per raccogliere e riportare la palla.
6. Scegli se "girarti" o "sdraiarti".
7. Aziona le parti del corpo per eseguire le azioni richieste.
8. Conta il numero delle volte che ti sei girato o quello dei secondi trascorsi durante l'esecuzione del compito.
9. Abbaia/Non abbaiare.
10. Fermati quando il compito è terminato.
11. Preparati a ricevere il prossimo compito.

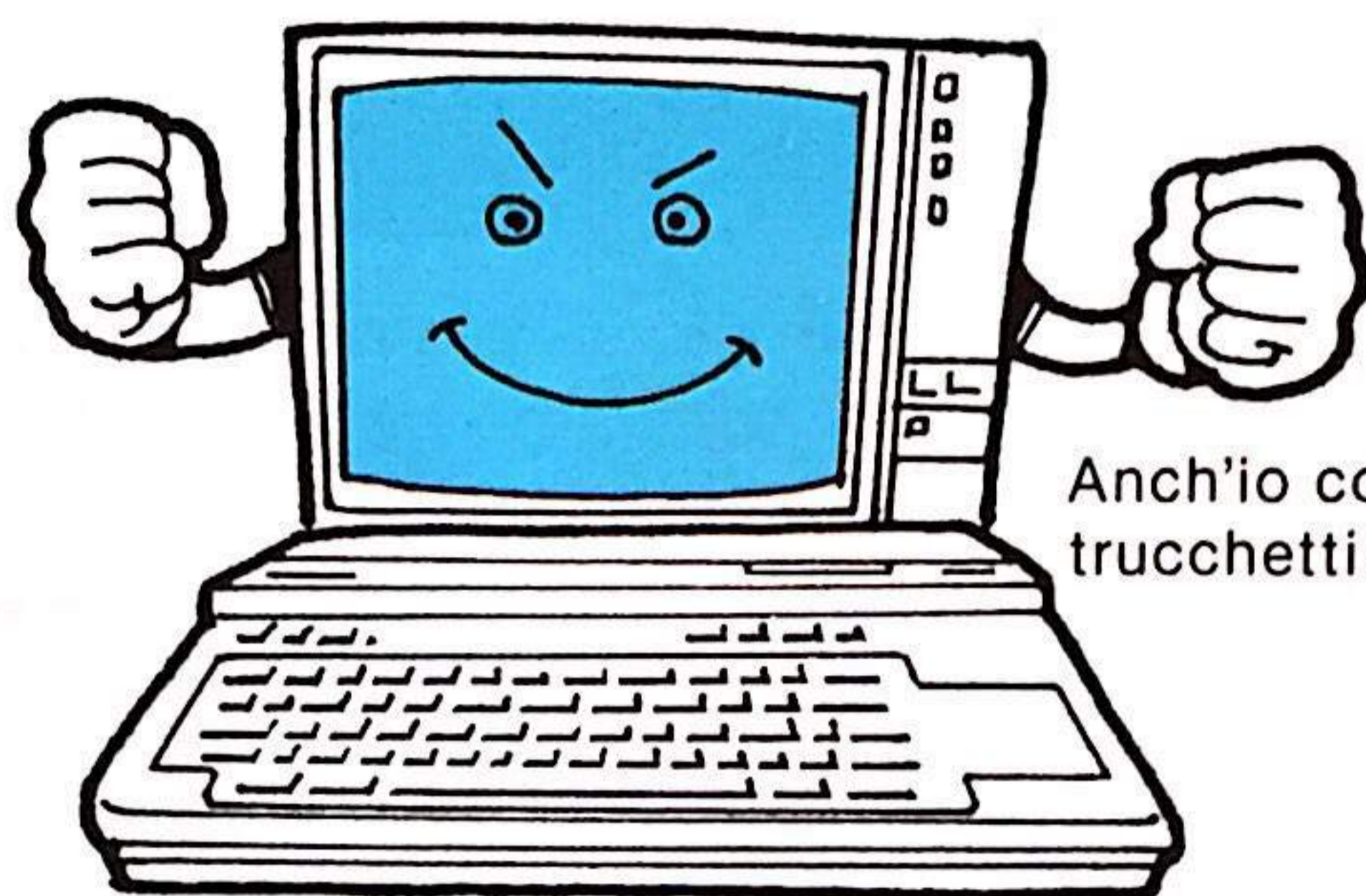
Dobbiamo controllare tutti questi passi per essere sicuri di non aver dimenticato nulla o di non aver fatto un errore, dopo di che possiamo introdurre il tutto nel microprocessore di Supercane. Se abbiamo pensato a tutto—ed elencato tutto nell'ordine corretto—Supercane si comporterà perfettamente ogni volta che dovrà eseguire un compito. Possiamo “eseguire il programma” una volta, dieci volte o diecimila volte, e lui farà sempre la stessa cosa.



Ci sono perciò due regole per scrivere un programma: pensare a **tutto** quello che il computer deve fare per portare esattamente a termine l'operazione che desideriamo e programmare tutti i passi nell'**ordine esatto**. L'unica “abilità” necessaria è il normale e quotidiano ragionamento logico, perché i computer, come del resto tutte le macchine, funzionano come funziona il nostro cervello—logicamente.

E ora, scriviamo il primo programma sul nostro Computer Sony. Vedrete come sarà facile.

SCRIVIAMO UN PROGRAMMA: FACILE COME UNO, DUE, TRE



Anch'io conosco dei bei trucchetti!

“Sequenza” in un programma per computer significa semplicemente che i comandi vengono scritti in ordine successivo, come uno, due, tre. Vi ricordate il compito affidato a Supercane? Era diviso in tre parti diverse, ciascuna indicata da un numero.

1. Trova la palla e riportamela!
2. Se la palla è bianca, girati tre volte e abbaia!
3. Se la palla è nera, sdraiati per dieci secondi!

Questi passi sono ordinati in senso logico: vogliamo che il cane ci riporti la palla prima di fare le altre cose; di conseguenza, il comando 1 deve venire per primo (i comandi 2 e 3 possono essere scambiati senza cambiare la sostanza del compito).

Introdurremo i nostri comandi nel computer nello stesso modo: li scriveremo sullo schermo nello stesso ordine in cui vanno eseguiti e assegneremo un numero a ciascuno di essi. Il primo “lavoro” che daremo al computer—il nostro programma—sarà quello di far apparire sullo schermo le parole “PAOLO E MARIA”.

```
10 PRINT "PAOLO"  
20 PRINT "E"  
30 PRINT "MARIA"
```

Abbiamo già imparato il comando PRINT che qui useremo tre volte—una per ciascuna delle tre parole che vogliamo stampare. La sequenza sarà naturalmente quella dell’ordine delle parole. L’unica differenza rispetto al modo in cui abbiamo usato i comandi PRINT precedentemente, è che ora ciascuno di essi comincia con un numero: 10, 20 e 30. Questo è molto importante: questi numeri dicono al computer il **comando successivo da eseguire** durante l’esecuzione di ogni passo del programma. In altre parole, quale ordine seguire.

Battiamo ora il primo comando: 10 PRINT "PAOLO". Non dimentiamo: alla fine bisogna battere una volta **RETURN**.

1 0 [] P R I N T [] " P A O L O " [] RETURN

```
10 PRINT "PAOLO"
■
```

Perfetto. Il nostro primo programma è entrato nel computer, il cursore si è spostato sulla riga seguente e noi siamo pronti ad introdurre gli altri due comandi, esattamente nello stesso modo:

```
10 PRINT "PAOLO"
20 PRINT "E"
30 PRINT "MARIA"
■
```

Ecco qui il nostro programma—quasi pronto per essere usato.

CORREGGIAMO UN PROGRAMMA: CERCHIAMO GLI ERRORI

Prima di "far girare" il nostro programma, dobbiamo fare un'ultima cosa importante: **controllare** che ogni parte del programma sia stata immessa correttamente. Ricordate che il Computer Sony è solo una macchina e sa leggere solo i comandi scritti in BASIC. Se c'è un errore—anche una sola lettera o uno spazio battuto scorrettamente—il comando non sembra più una parola BASIC (o sembra una parola sbagliata). Un errore di questo tipo interrompe tutto il programma o fa fare al computer la cosa sbagliata. Per esempio:

```
10 PRIMT "PAOLO"
      ↑
      |
      |----- Errore
```

Errori come questo nel programma vengono chiamati **malfunzionamenti**. Il comando PRINT va battuto correttamente o il computer non capirà nemmeno che gli stiamo dando un comando.

Se troviamo un errore nel programma ora, non dobbiamo preoccuparce-ne: è meglio trovarlo ora che dopo ed è molto facile correggerlo.

Abbiamo già imparato a muovere il cursore usando i quattro tasti con su scritta una freccia. Il cursore si trova ora sotto il programma, perciò, usando l'apposito tasto, dobbiamo portarlo in su, in corrispondenza di "10". Quindi, con il tasto relativo, spostiamo il cursore di sei spazi a destra, dove si trova la lettera sbagliata M.

```
10 PRINT "PAOLO"
      ↑
      |
      |----- Portiamo qui il cursore
```

Per correggere, basta battere la lettera corretta.

```
10 PRINT "PAOLO"
      ↑
      |
      |----- Premiamo [N]
```

Ora, premiamo una volta [RETURN] e l'errore non c'è più.

```
10 PRINT "PAOLO"
20 PRINT "E"
      ↑
      |
      |----- Dopo aver premuto [RETURN]
                il cursore si sposta qui.
```

Se ci sono degli altri errori, correggeteli muovendo il cursore esattamente nello stesso modo. (Se ce ne fosse bisogno, potete controllare il **Manuale delle istruzioni** per imparare ad aggiungere un carattere o uno spazio che avete dimenticato, o per togliere un carattere che non è necessario). Dopo aver corretto tutti gli errori, usate i tasti contrassegnati dalla freccia per riportare il cursore in fondo al programma—in modo che lo schermo abbia di nuovo questo aspetto:

```
10 PRINT "PAOLO"
20 PRINT "E"
30 PRINT "MARIA"
■
```

Può darsi che siamo riusciti ad eseguire questo semplice programma senza fare nessun errore. È tuttavia molto importante **controllare** ogni volta. Col passare del tempo i nostri programmi conterranno più comandi, molti comandi diverranno più lunghi. Qualsiasi errore, grande o piccolo, può interrompere un programma e **tutti**—anche gli esperti della Sony—fanno errori di battitura ogni tanto.

Una cosa è certa però: se c'è qualche errore, il nostro computer Sony prima o dopo ce lo dirà. Un **messaggio di errore** apparirà sullo schermo, il programma si fermerà prima della fine, oppure il computer si comporterà diversamente da come volevamo. In tal caso dovremo controllare l'intera lista dei comandi, localizzare l'errore e correggerlo. Perciò è meglio controllare e "correggere" il programma fin dall'inizio, già al momento di battere i comandi.

Regole per la programmazione

- Pensare a **ogni passo che sarà necessario al programma** per far fare al computer quello che si desidera
- Accertarsi di elencare tutti i comandi—nella **sequenza logica** che porterà al risultato corretto
- Usare sempre un **numero di sequenza** all'inizio di ogni riga di comando
- Dopo aver introdotto i comandi, **controllarli attentamente uno per uno** cercando eventuali errori—e correggere quelli che si trovano

FACCIAMO GIRARE IL PROGRAMMA

Ora che abbiamo immesso tutti i comandi e fatto le correzioni necessarie, siamo pronti a far girare (**RUN**) il programma. RUN è perciò il comando che facciamo apparire sullo schermo:

RUN

E questo è quello che il computer fa con il nostro programma dopo che lo abbiamo immesso e abbiamo premuto il tasto **RETURN**. Questo è ciò che appare ora sullo schermo:

```
10 PRINT "PAOLO"  
20 PRINT "E"  
30 PRINT "MARIA"  
RUN  
PAOLO  
E  
MARIA  
OK  
■
```

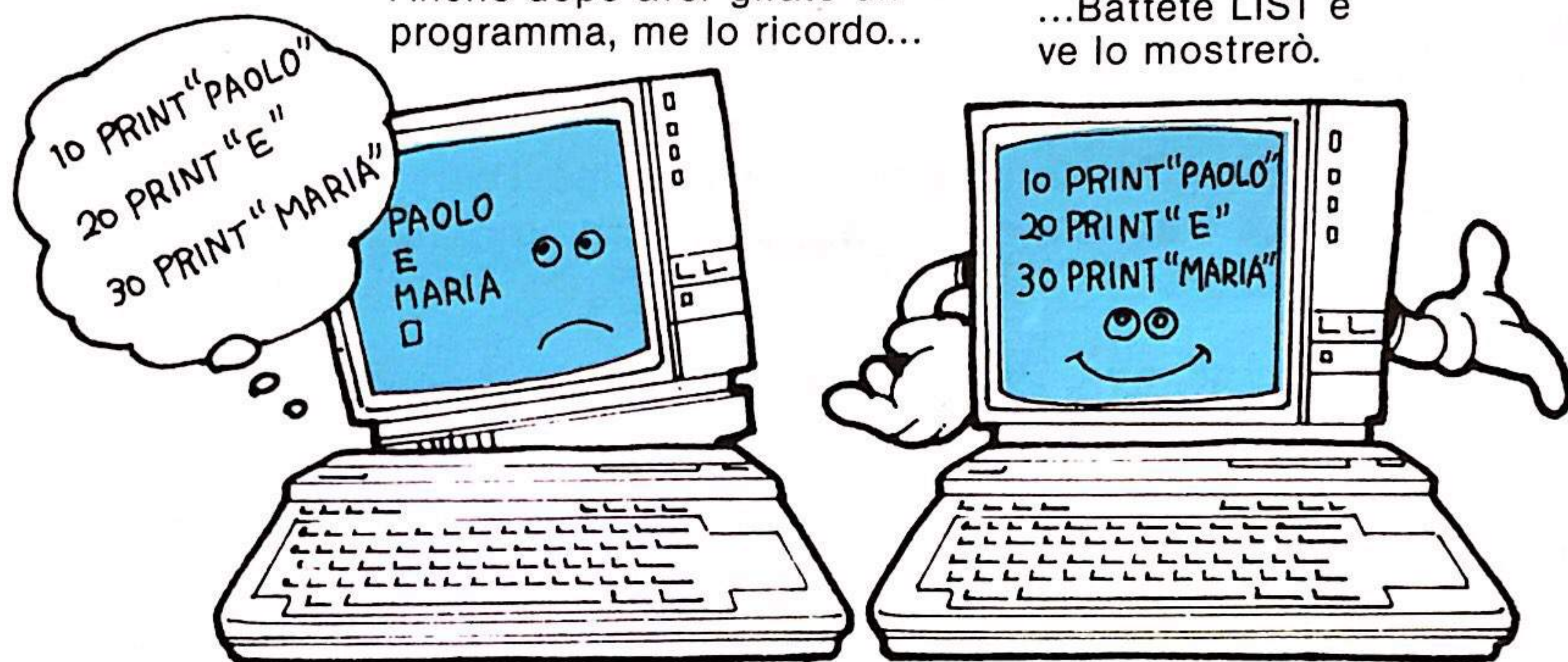
Il nostro Computer Sony ha impiegato solo alcuni secondi per "leggere" i comandi e stamparli.

Battiamo di nuovo il comando RUN, subito dopo il cursore, e premiamo di nuovo **RETURN**. Il risultato è lo stesso, no? Il risultato di **questo** programma sarà sempre lo stesso, fino a quando non cambieremo o cancelleremo i comandi. Possiamo provare quante volte vogliamo per rendercene conto. Le parole PAOLO E MARIA appariranno ogni volta. Questo significa che il Computer Sony **ricorda** il programma.

Possiamo controllare la memoria in qualsiasi momento immettendo il comando **LIST** (e premendo poi **RETURN**). Useremo il **comando LIST** ogni volta che scriveremo un programma.

Niente paura!
Anche dopo aver girato un programma, me lo ricordo...

...Battete LIST e ve lo mostrerò.



La cosa più utile del nostro Computer Sony è che ricorda i comandi che gli diamo. Tutto quello che dobbiamo dirgli è di ricordarli, mettendo un **numero davanti al comando**.

```
10 PRINT "PAOLO"
```

↑
Se si immette un numero di riga, è un programma.

Siccome questo programma è preceduto da un numero, il computer sa che fa parte di un programma e lo memorizzerà. **10** è il **numero di riga** e un numero di riga può essere un qualsiasi numero da 0 a 65529.

Per quanto tempo il computer può ricordare i nostri comandi? Per sempre!—o fino a quando non gli diciamo di dimenticarsene. Il comando per far dimenticare un precedente comando al computer è

NEW

Assicuriamoci che il computer si ricordi di un comando ricevuto, fornendogli alcuni comandi numerati e introducendo poi LIST. Il computer li ha listati? Perfetto. Ora battiamo NEW.

```
LIST
10 PRINT "PAOLO"
20 PRINT "E"
30 PRINT "MARIA"
OK
NEW
OK
■
```

Sullo schermo sono ancora visibili i comandi ma il computer li ha dimenticati. Proviamo a battere di nuovo LIST. Il programma è uscito dalla memoria del computer.

Ci sono due modi per far sì che il computer dimentichi i comandi ricevuti. Uno consiste nel premere il tasto **RESET** e l'altro nello spegnere il computer. **Non fate nessuna di queste due cose** a meno che non siate sicuri di volere che il computer dimentichi i comandi.



In seguito impareremo a **conservare** i programmi, in modo da poterli riusare settimane, o anche anni, dopo. Fino ad allora, il computer li dimenticherà ogni volta lo spegneremo.

■ GRAFICI: STELLE LUMINOSE ■

Come...

- Usare altre parole BASIC
- Ripulire lo schermo
- Disegnare punti e linee
- Far ripetere un'azione al computer
- Usare una variabile
- Usare un numero casuale
- Usare i nuovi comandi COLOR



UN PROGRAMMA DI GRAFICI

Proviamo ad usare alcune nuove parole BASIC e cerchiamo di imparare a che cosa servono:

```
10 SCREEN 2
20 PSET (100,100)
30 PSET (150,100)
40 PSET (100,150)
50 PSET (150,150)
60 GOTO 20
```

Questo è un **programma**—un insieme di passi che il computer può usare per raggiungere un qualche scopo. Ora che il programma è stato memorizzato, proviamo a dare il comando RUN e vediamo che cosa combina questo programma.

Tutte le lettere sono scomparse e al loro posto sono apparsi sullo schermo quattro punti bianchi. Anche il cursore è scomparso e niente appare sullo schermo quando proviamo a premere un tasto, non è vero? Il computer è occupato a far girare il programma. Perché ci mette tanto?

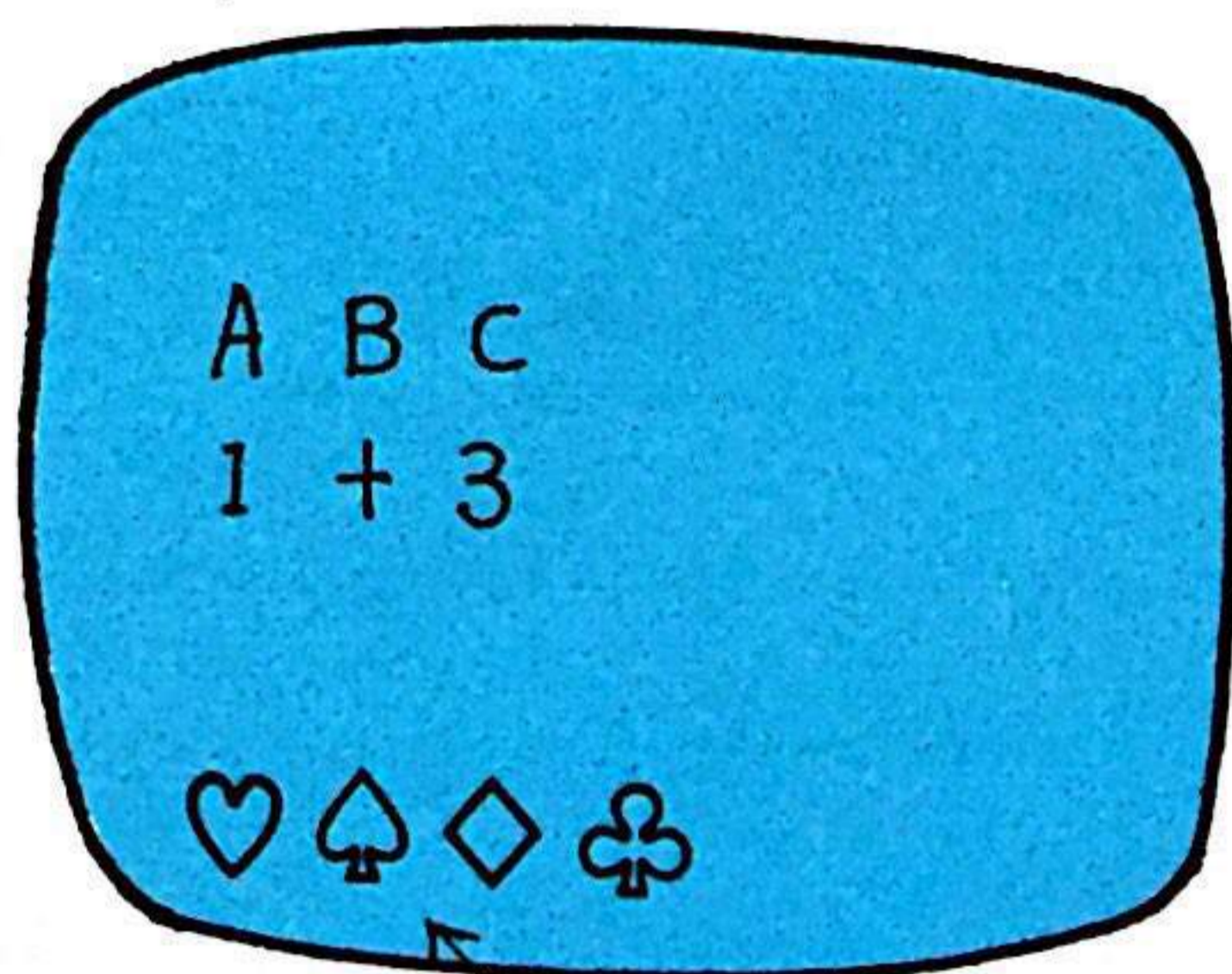
Proviamo a guardare che cosa significa questo programma. Prima di tutto, premiamo **CTRL** e **STOP** per fermare il programma. I punti sono scomparsi e il cursore è tornato al suo posto. Battiamo ora LIST e osserviamo il programma riga per riga. Diamo un'occhiata prima alla riga 10.

SCREEN 2

è il comando che ci permette di cominciare a disegnare **grafici** (immagini) sullo schermo. I grafici sono costituiti da punti, linee, cerchi e altre figure e disegni che vengono visualizzati sullo schermo (Le figure che vengono visualizzate sullo schermo non sono grafici—numeri, lettere e simboli—e sono chiamate **caratteri**).

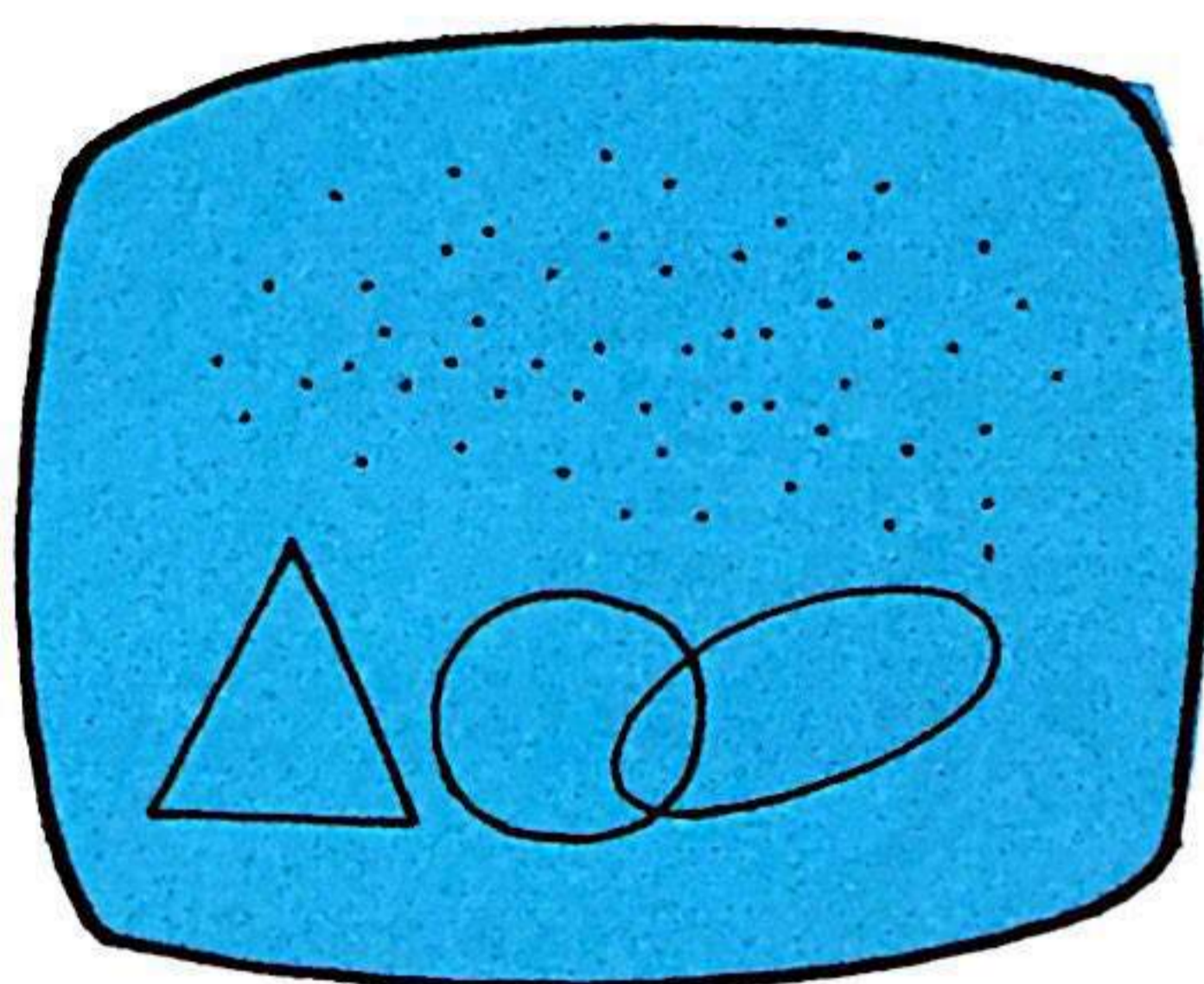
I comandi SCREEN dicono al computer la forma in cui l'informazione deve essere visualizzata sullo schermo. Ci sono due tipi di comandi SCREEN.

SCREEN 0 o SCREEN 1
per i **caratteri**



(I simboli sulla tastiera sono caratteri)

SCREEN 2 per i **grafici**



Il comando SCREEN 2, quindi, ordina al computer di far apparire dei grafici. Per questo motivo, quando abbiamo battuto tale comando, i caratteri visualizzati sullo schermo sono scomparsi. Premendo **CTRL** e **STOP** invece, abbiamo annullato il comando dei grafici e riportato i caratteri sullo schermo. Ora che abbiamo capito la prima riga del programma possiamo passare alla seguente.

```
20 PSET (100,100)
30 PSET (150,100)
40 PSET (100,150)
50 PSET (150,150)
```

Spiegazione: il comando PSET serve a far comparire dei **punti** sullo schermo e i numeri contenuti tra le parentesi () dicono al computer dove sistemare tali punti.

Ripeti, ripeti, ripeti, ripeti

L'ultima riga del programma contiene un comando molto importante:

```
60 GOTO 20
```

GOTO significa "va'a", il che significa che questo comando dice al computer di andare indietro alla riga 20 e di ricominciare da capo. Quando arriva alla riga 20,

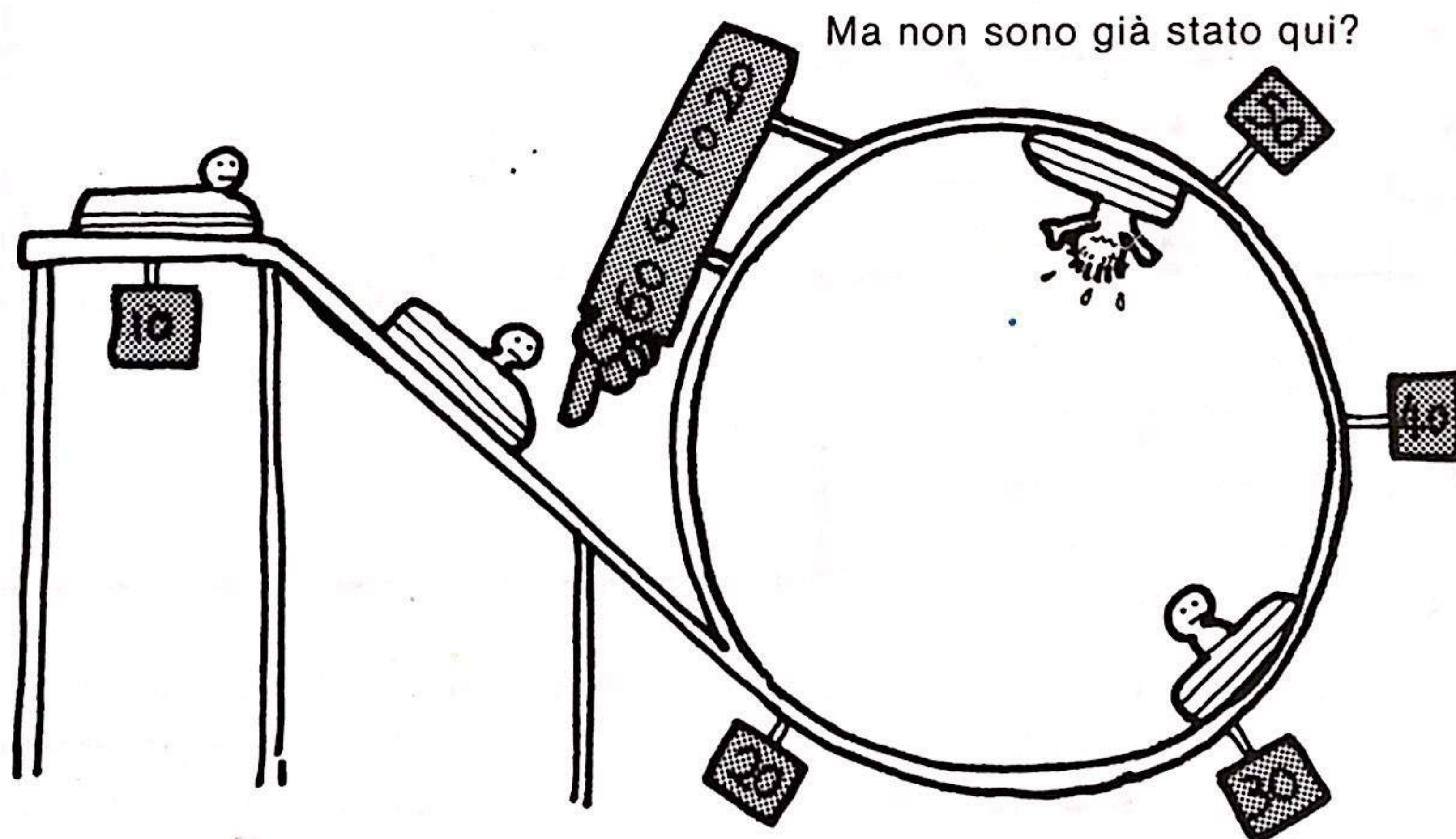

```
20 PSET (100,100)
```

il computer ripete il programma. Quando arriva alla riga 60, ritorna alla riga 20, ripete il programma e quindi torna alla riga 20, ripete il programma e quindi ...

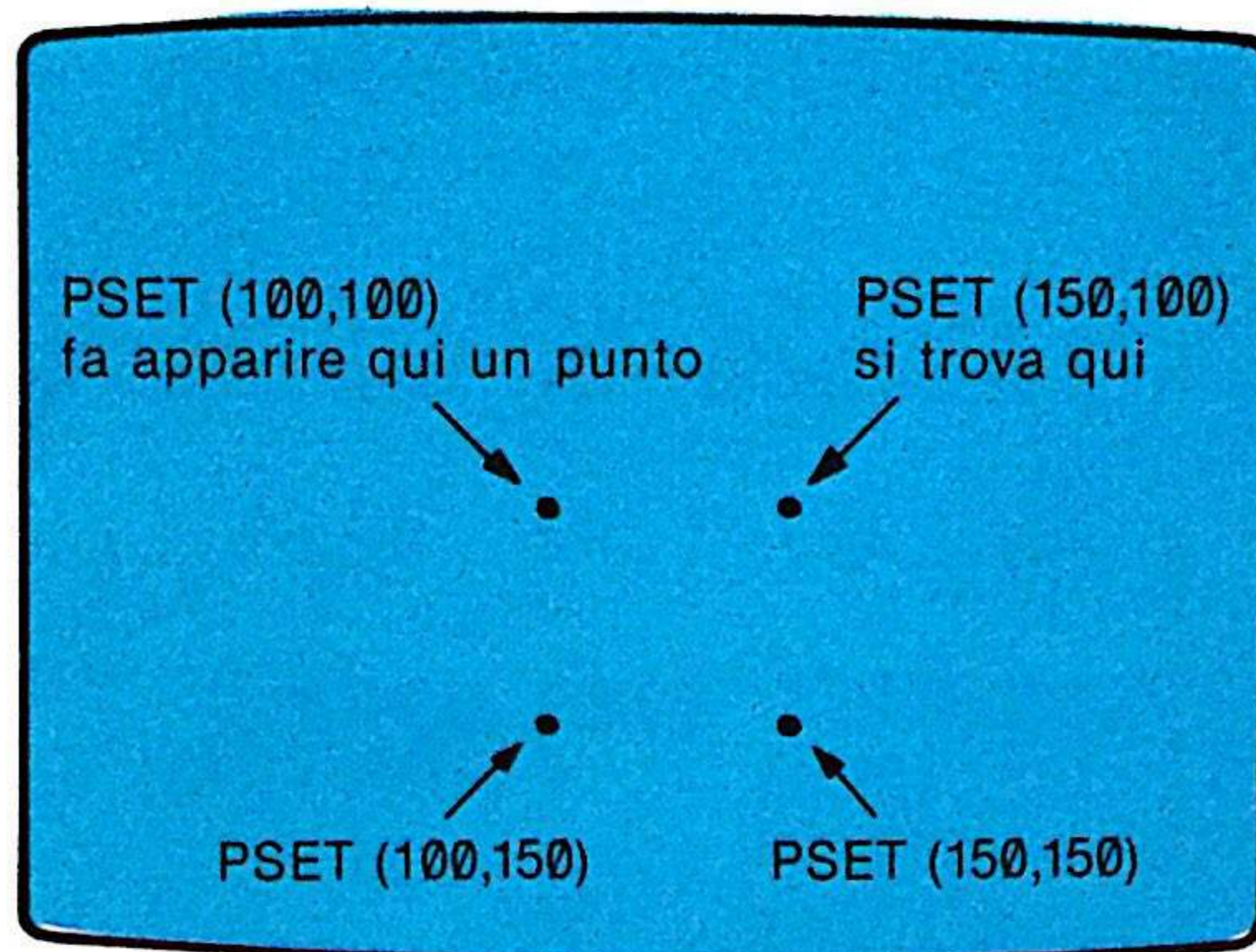
```
10 SCREEN 2
↓
20 PSET (100,100)
↓
30 PSET (150,100)
↓
40 PSET (100,150)
↓
50 PSET (150,150)
↓
60 GOTO 20
```

Questa parte viene ripetuta all'infinito

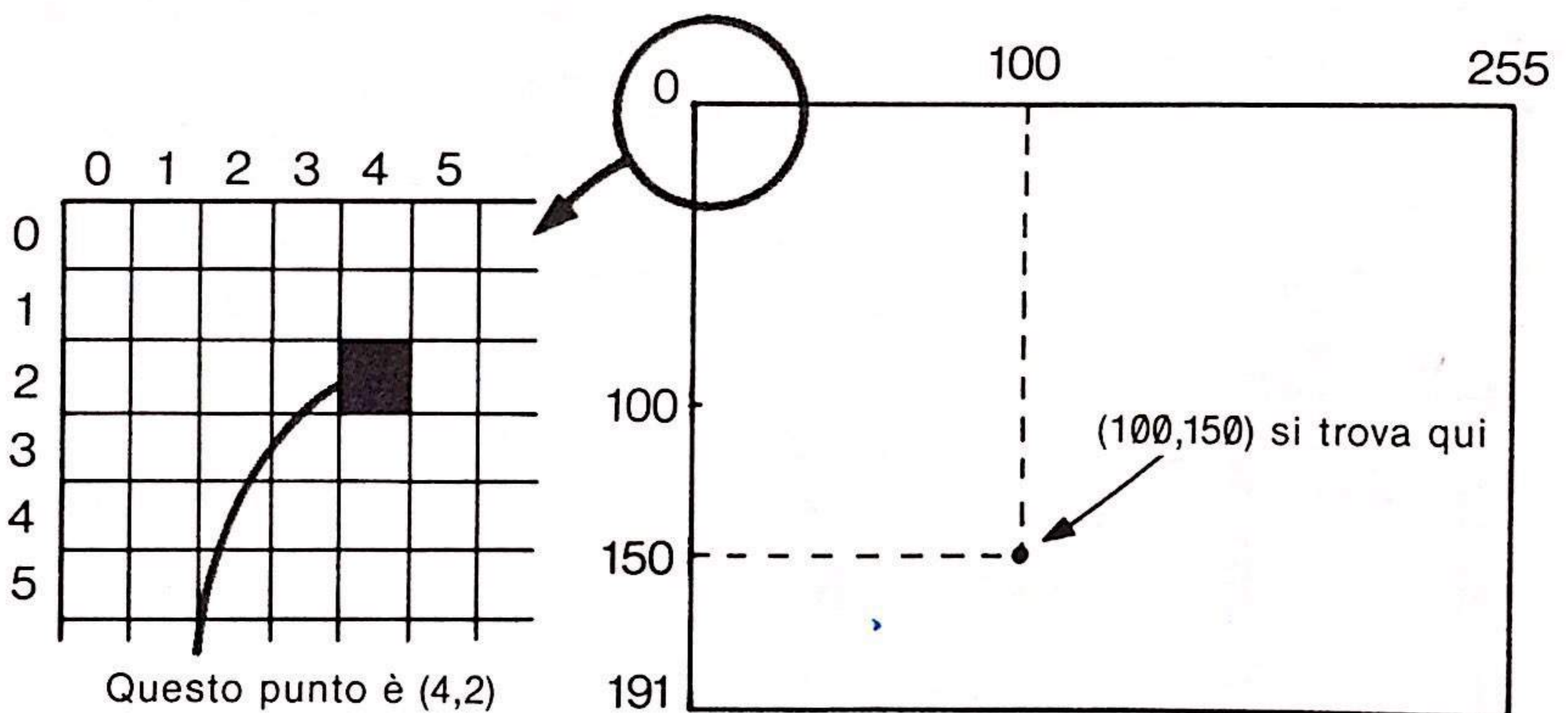
Una sezione di programma che viene ripetuta si chiama **ciclo**, un espediente comune e molto utile nella programmazione dei computer. (Per fortuna, non tutti i cicli si ripetono all'infinito).



Vediamo in che modo quei comandi PSET fanno apparire i punti sullo schermo.



Avete osservato come funzionano i numeri? Per entrambi i numeri che si trovano sulla destra il primo numero è 100, mentre per i due numeri che si trovano sulla sinistra il numero è 150. Ecco un'immagine ingrandita dello schermo.



Per dire al nostro Computer Sony dove sistemare un punto, dobbiamo fornirgli due numeri di posizione, il primo per la posizione sinistra/destra (orizzontale) e il secondo per la posizione alto/basso (verticale). Sullo schermo sono disponibili 256 diverse posizioni da sinistra a destra (da 0 a 255) e 192 dall'alto al basso (da 0 a 191). Possiamo perciò sistemare un punto in qualsiasi posizione dello schermo che si trovi tra (0,0) e (255,191).

Grafici e variabili

Come abbiamo visto, qualsiasi numero può essere sostituito da una variabile. Questo vale anche per i numeri PSET. Se trasformiamo il numero di posizione sinistra/destra nella variabile X e quello di posizione alto/basso nella variabile Y, il nostro comando diventerà

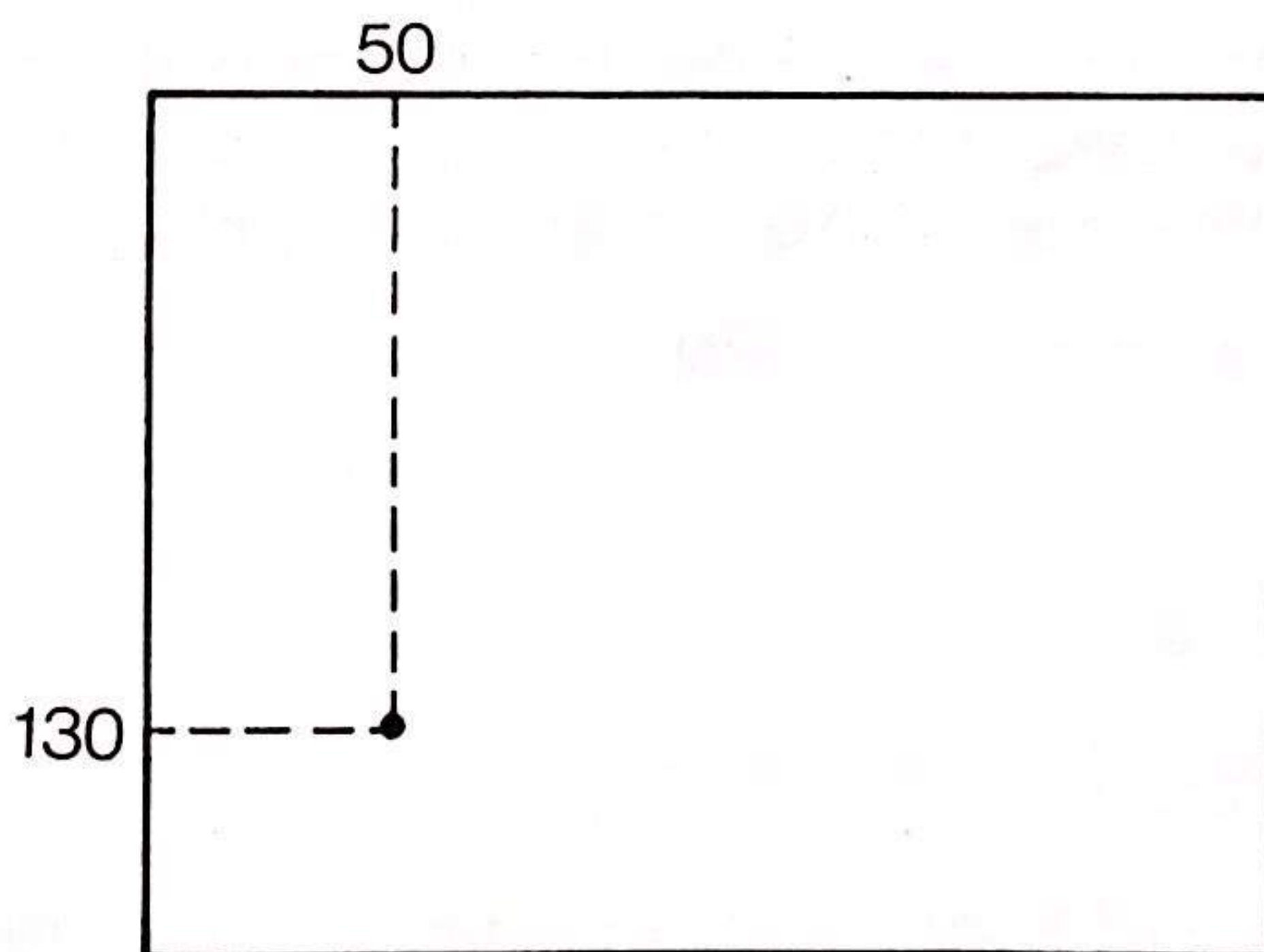
PSET (X,Y)

e potremo usare qualsiasi numero vogliamo per X e Y e far muovere i punti a nostro piacere.

Come si usano le variabili per un singolo punto? Prima, battiamo il comando NEW per far dimenticare al computer l'ultimo programma e poi introduciamo nel computer questo nuovo programma:

```
10 SCREEN 2
20 X=50:Y=130
30 PSET (X,Y)
40 GOTO 30
```

Premiamo il tasto **RETURN** per memorizzare il programma e quindi diamo il comando RUN. L'esecuzione del programma inizia dalla riga 10 dove lo schermo viene liberato e preparato per i **grafici**. Alla riga 20 il computer impara due variabili che usa alla riga 30 per sistemare un punto sullo schermo.



Siccome X vale 50 e Y vale 130, un punto bianco compare sullo schermo nella posizione (50, 130). Il computer legge quindi la riga 40—che lo rimanda alla riga 30. Ciò significa che l'unico modo per interrompere questo programma ciclico è premere i tasti **CTRL** e **STOP**.

Perché abbiamo usato delle variabili quando sarebbe stato più semplice battere PSET (50, 130)? E perché abbiamo costretto il programma a ripetersi all'infinito con la riga 40? In effetti, questo procedimento non è necessario se ci accontentiamo di far apparire un solo punto sullo schermo per un breve periodo. Tale procedimento è però molto comodo quando vogliamo far apparire molti punti luminosi in un programma molto breve. Questo è proprio quello che faremo tra poco, dopo aver letto questa breve nota sulla punteggiatura e sugli errori di battitura.

Nota: Stiamo attenti alla punteggiatura

Il programma che stiamo usando ora contiene tre tipi diversi di **punteggiatura**—la , (virgola), i : (due punti) e le () (parentesi). Questi segni di punteggiatura sono considerati **caratteri** e **vanno sistemati al posto giusto**, come le lettere delle parole di comando. Ricordate che il computer Sony non può capire un comando se questo non è scritto esattamente e questo riguarda anche la punteggiatura.

Ogni segno di punteggiatura ha un significato particolare nel BASIC. La virgola indica che un numero è terminato e che ne inizia un altro. I due punti indicano che un comando è terminato e che subito dopo, sulla stessa riga, ce ne sarà un altro. Le parentesi sono necessarie quando due o più numeri, o due o più variabili, vengono usati insieme—e **vanno sempre in coppia**, prima la sinistra (“e poi la destra”). Impareremo più avanti alcuni degli altri segni di punteggiatura che vengono usati nei comandi BASIC.

Come abbiamo detto prima, chiunque può fare degli errori di battitura. Quando questo succede, il computer emette un segnale di errore perché non è in grado di capire il comando. Per esempio,

```
Syntax error in 30
```

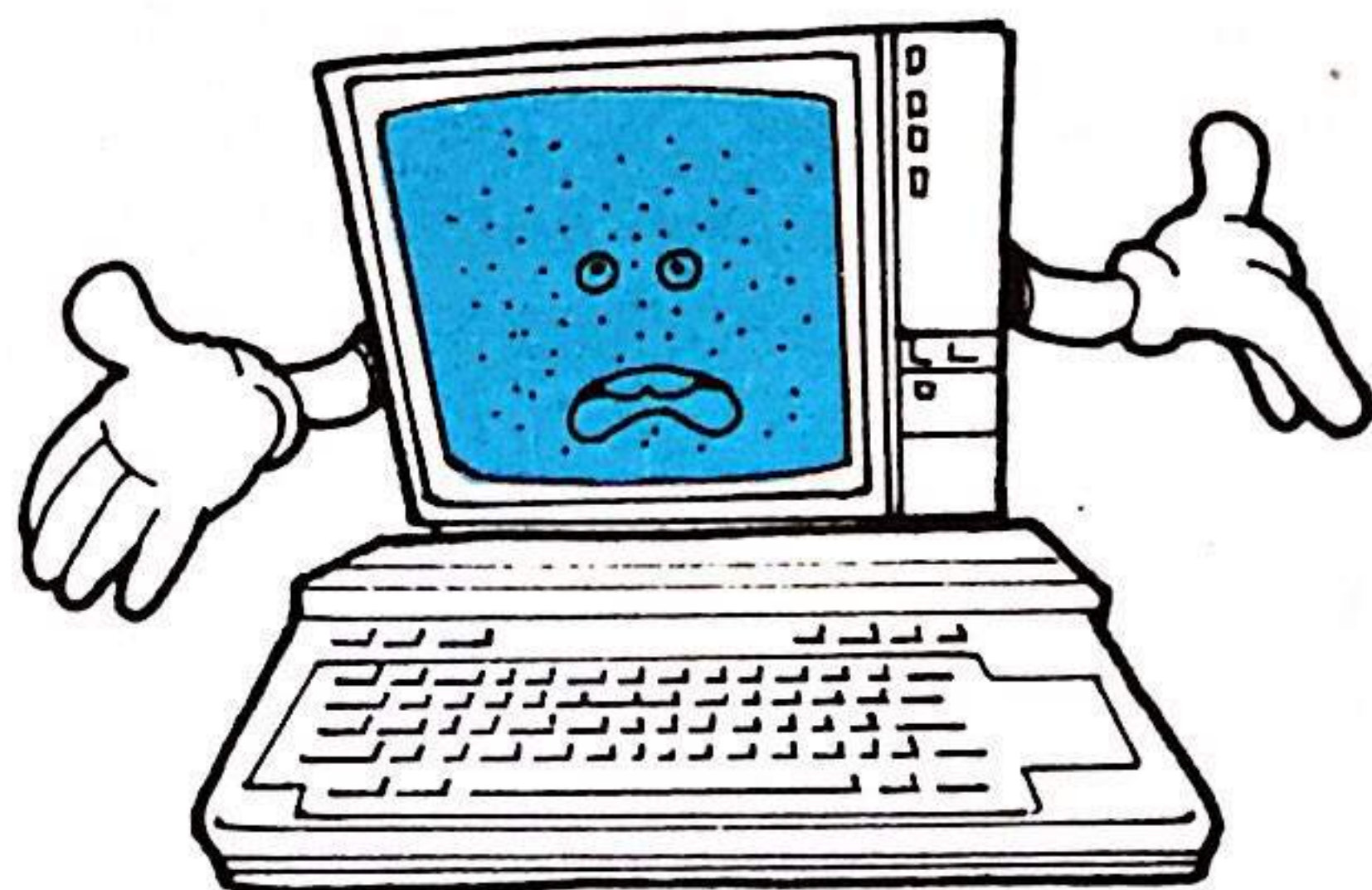
significa che probabilmente c'è un errore alla riga 30.

NUMERI CASUALI

Usiamo il comando NEW per ripulire la memoria del computer e battiamo questo nuovo programma:

```
10 SCREEN 2
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
40 PSET (X,Y)
50 GOTO 20
```


Durante l'esecuzione del programma lo schermo andrà riempiendosi sempre più di punti, fino a quando non lo fermeremo ... proprio così, con i tasti **CTRL** e **STOP**.



Che cosa combini con la mia faccia?

E ora spieghiamo queste due nuove righe del programma.

```
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
```

Non è difficile capire che queste due righe assegnano dei valori a X e a Y. Ma cosa significano questi valori? Consideriamo prima RND(1). RND significa **casuale** ed è una delle **funzioni** del BASIC. Proviamo a usare questo comando per capire la **funzione dei numeri casuali**.

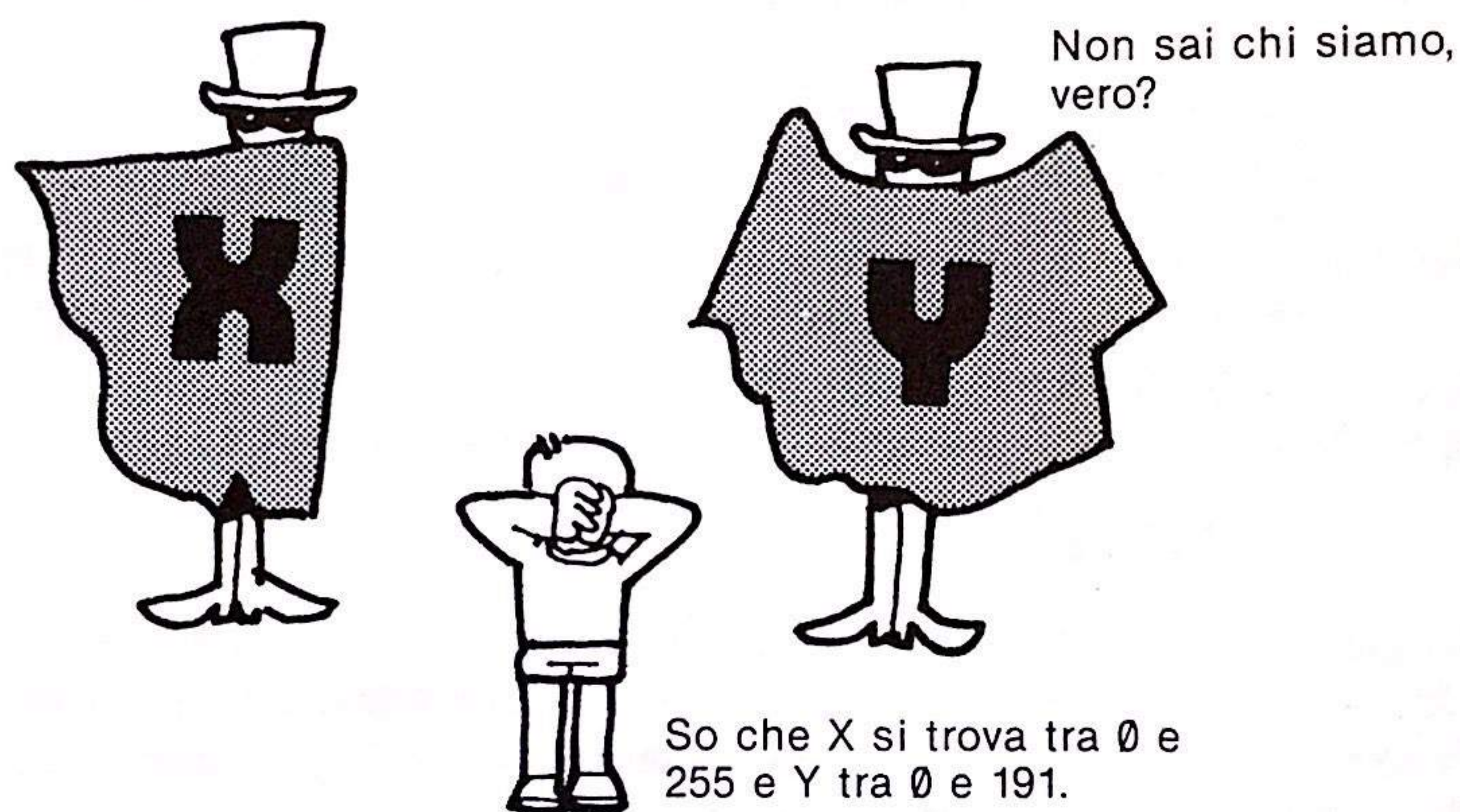
```
X=RND(1):PRINT X
```

Dopo aver immesso questo comando battiamo **RETURN**: sullo schermo apparirà una frazione a 14 cifre, un numero minore di 1 ma maggiore di zero che comincia con la **virgola decimale** (in BASIC la virgola decimale si scrive come un punto). Per esempio:

```
X=RND(1):PRINT X
.59521943994623
OK
■
```

Riproviamo a usare questo comando. È cambiato il numero? Riproviamo. Il computer possiede una lunga lista di numeri casuali e li sceglie uno dopo l'altro per assegnarli alla X.

Se torniamo alla riga 20 del programma troviamo $(\text{RND}(1) * 256)$, che significa che il numero casuale viene moltiplicato per 256. Tuttavia, se moltiplichiamo quelle frazioni per 256, anche il risultato conterrà delle frazioni—mentre le posizioni di ogni “punto” devono essere costituite da numeri **interi**, come 1, 30 o 192, perché i numeri di posizione sullo schermo sono fissati solo in numeri interi. Per questo motivo il comando inizia con INT. INT significa **intero**, o numero intero, ed elimina la virgola decimale e i numeri che la seguono, trasformando ogni risultato in un numero intero. Ora il significato di X è chiaro. Cambia ogni volta che il programma ripete il suo ciclo. Ogni volta che il computer legge $X = \text{INT}(\text{RND}(1) * 256)$, assegna a X un numero intero arrotondato tra 0 e 255—senza che si possa prevedere quale numero sarà. Non solo, ogni volta che il computer legge la riga 40, assegna ad Y un numero intero compreso tra 0 e 191, esattamente nello stesso modo.



Abbiamo già capito che cosa significa la riga seguente

```
40 PSET (X,Y)
```

anche se non sappiamo ancora quali saranno i valori di X e Y. Il computer non ci dirà quali sono questi valori perché non ha ricevuto il comando PRINT. Continuerà, invece, a distribuire per lo schermo i punti, nelle posizioni che gli verranno indicate dai numeri interi delle posizioni sinistra/destra e alto/basso.

Finora abbiamo spiegato solo come un punto va a finire sullo schermo. Ma da dove vengono tutti gli altri? È proprio questa la funzione del ciclo, creare tutti questi punti. La riga seguente

```
50 GOTO 20
```

fa tornare il programma a


```
20 X=INT(RND(1)*256)
```

Ogni volta vengono assegnati dei numeri nuovi a X ed Y di modo che

```
40 PSET (X,Y)
```

mette ogni punto in un posto diverso e noi vediamo i punti apparire l'uno dopo l'altro e distribuirsi in giro per lo schermo.

Programmi: due tipi di comandi

Come abbiamo detto, un programma è **una serie di comandi in ordine fisso e numerato** immessa nella memoria del computer. Un programma può contenere centinaia o migliaia di righe—comandi—ma un numero così grande non è necessario nella maggior parte dei casi; con un programma di sole cinque righe, abbiamo appena fatto apparire sullo schermo migliaia di punti.

Una di quelle cinque righe—GOTO 20—non ha fatto fare niente al computer, ma gli ha detto piuttosto **dove andare dopo**. È questa la riga che ha permesso al nostro breve programma di realizzare così tanto—fabbricando un ciclo che si ripete all'infinito.

Possiamo così vedere che esistono due tipi di comandi:

1. Comandi che dicono al computer cosa fare, come:

PRINT PSET WIDTH COLOR etc.

e

2. Comandi che cambiano l'ordine del programma

GOTO IF—THEN FOR—NEXT etc.

I comandi del secondo tipo sono poco numerosi e li ritroveremo più avanti. Sono forse gli strumenti più interessanti lavorando con un computer: come GOTO, quello che abbiamo usato per fabbricare un ciclo, vengono tutti usati per rendere **molto potenti** dei **piccoli** programmi.

METTIAMO DEL COLORE NEI NOSTRI PROGRAMMI

E ora, cerchiamo di rendere più divertente il nostro programma di punti—aggiungendovi una nota di colore! Per cominciare, con il comando LIST facciamo apparire sullo schermo il programma contenuto nella memoria del computer (nel caso avessimo spento il computer, dovremo rifare il programma).

```
LIST
10 SCREEN 2
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
40 PSET (X,Y)
50 GOTO 20
```

Poi, cambiamo la riga 40 in: 40 PSET (X,Y),2

```
10 SCREEN 2
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
40 PSET (X,Y)■
50 GOTO 20
```

Portiamo il cursore in questa posizione e poi battiamo ,2 (non dimenticate la virgola—è importante). Premiamo una volta **RETURN** per immettere il tutto e portiamo di nuovo il cursore sotto il programma. Diamo ora il comando RUN.

Da bianchi, i punti sono diventati verdi perché 2 è il numero di codice del colore verde medio. Usando la tabella dei colori di pagina 12, possiamo scegliere altri colori, se vogliamo. Oppure, possiamo trasformare il colore in una variabile e lasciare che il computer cambi a piacere il colore dei punti.

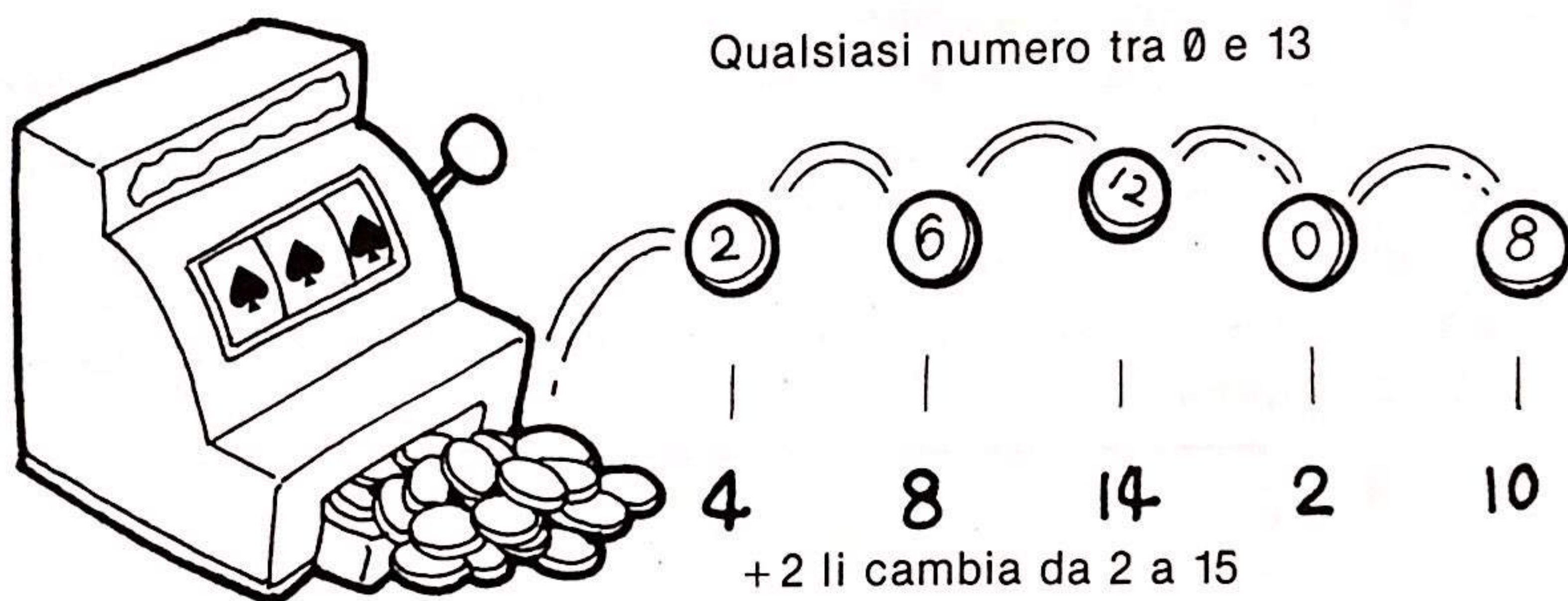
Torniamo alla riga 40 e cambiamo il codice del colore nella variabile C.

```
10 SCREEN 2
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
40 PSET (X,Y),C
50 GOTO 20
```


Ora, dobbiamo assegnare un valore alla variabile C, dare cioè il comando per il colore che desideriamo. È più divertente però se anche stavolta lasceremo al computer il compito di decidere il valore, a caso, come prima. Questa è la riga del comando che ci serve—ma aspettiamo un momento prima di immetterla nel computer.

```
35 C=INT(RND(1)*14)+2
```

Cosa significa? È un'istruzione con numero casuale come quelle che abbiamo visto prima per X e Y, con un passo in più alla fine. Sceglie un numero a caso tra 0 e 13 e vi aggiunge 2 (+2). Ciò significa che il numero sarà compreso tra 2 e 15. Ci sono però 16 colori che vanno dallo 0 al 15. Perché questa istruzione non comprende i colori con codice 0 e 1? Perché 0 è trasparente e i punti trasparenti sono invisibili e 1 è nero, che è ugualmente invisibile se anche lo sfondo è nero.



In quale punto del programma vogliamo mettere questa riga? Va messa prima che i punti vengano visualizzati a partire dalla riga 40—così questa nuova riga diventa la numero 35 (ecco perché le righe di istruzione vengono numerate per 10, 20, 30 ..., per permetterci di inserire dei nuovi comandi tra quelli che abbiamo precedentemente scritto). E ora immettiamo il programma.

```
10 SCREEN 2
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
40 PSET (X,Y),C
50 GOTO 20
35 C=INT(RND(1)*14)+2
■
```

30, 40, 50, 35. Pensate che il computer capirà? È naturale che capirà, perché sa contare e dispone automaticamente le righe in ordine numerico. È facile controllare se ha capito giusto—basta immettere il comando LIST.


```

LIST
10 SCREEN 2
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
35 C=INT(RND(1)*14)+2
40 PSET (X,Y),C
50 GOTO 20

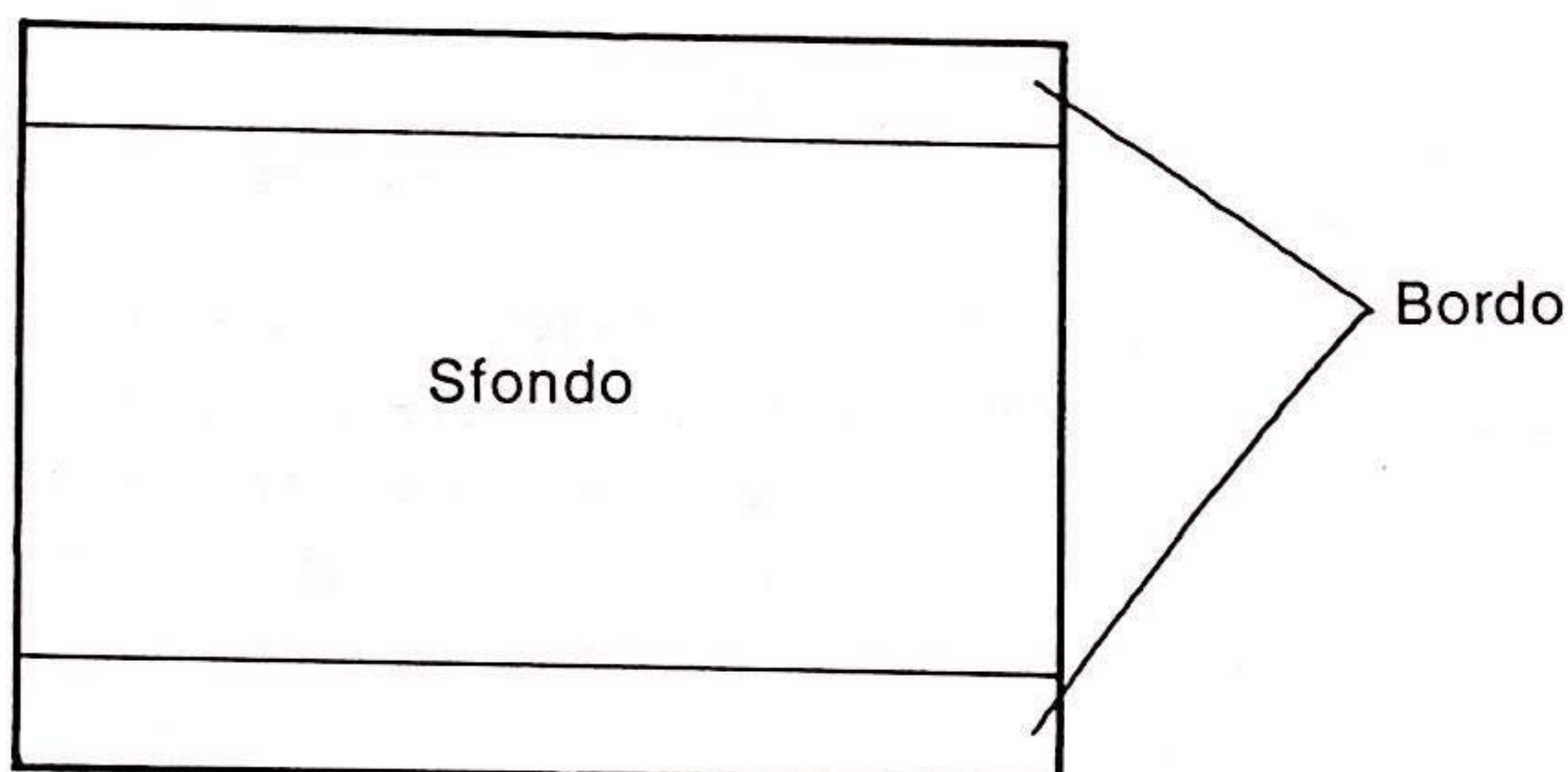
```

È giunto ora il momento di trasformare i nostri punti in **stelle**. Siccome le stelle si vedono solo di notte, bisogna annerire lo schermo. Immettiamo questa nuova riga:

```
5 COLOR 15,1,1
```

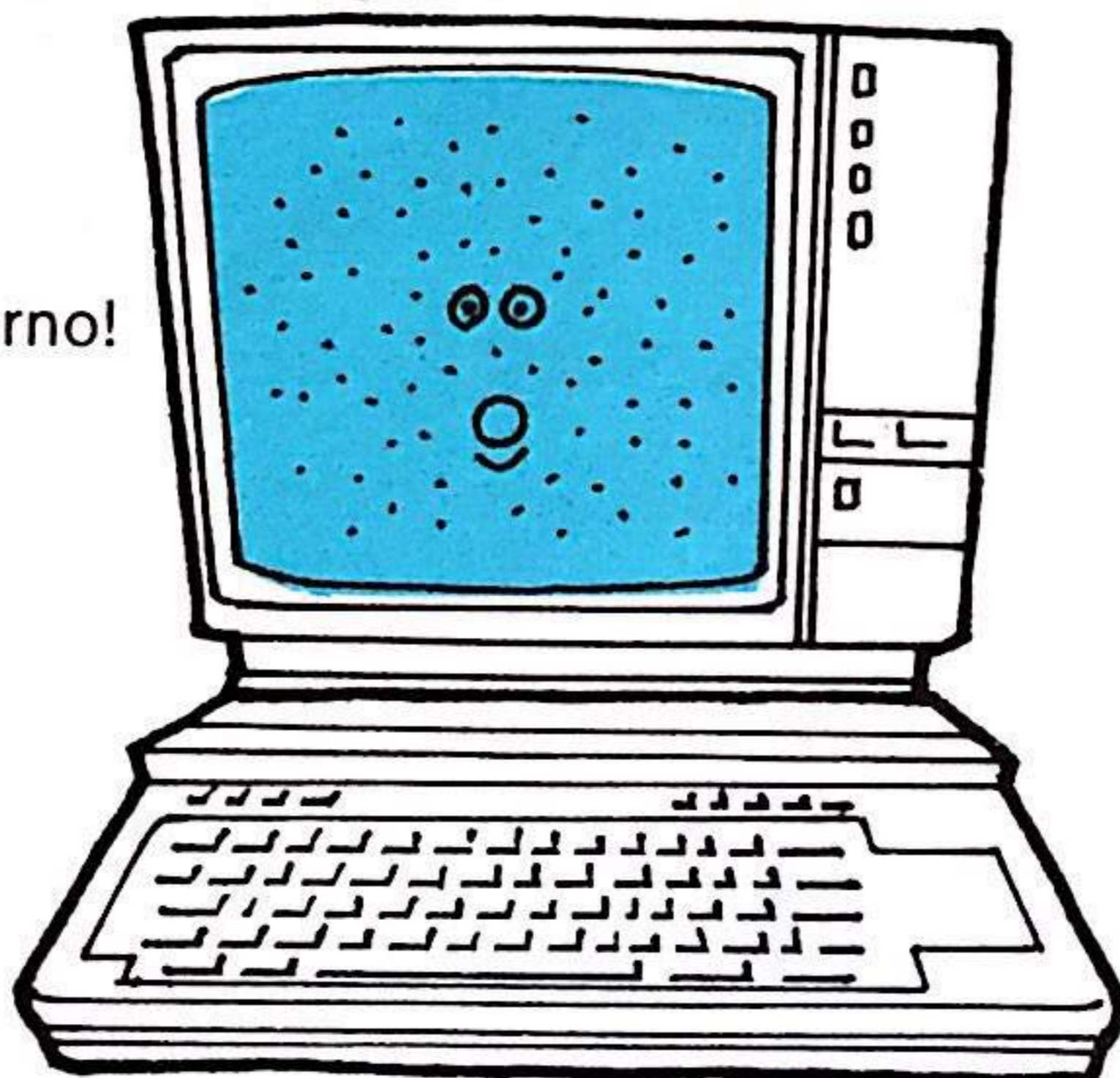
Nuovi comandi per colori e righe

Con questa riga impareremo un modo nuovo di usare il comando COLOR. Questo comando possiede tre codici: 15 stabilisce il colore del **carattere** (o il colore in primo piano), il primo 1 stabilisce il colore dello **sfondo** e l'ultimo 1 stabilisce il colore del **bordo** (Questa volta lo sfondo e l'area esterna avranno lo stesso colore. Si può usare qualsiasi numero, ma cominciamo con questi).

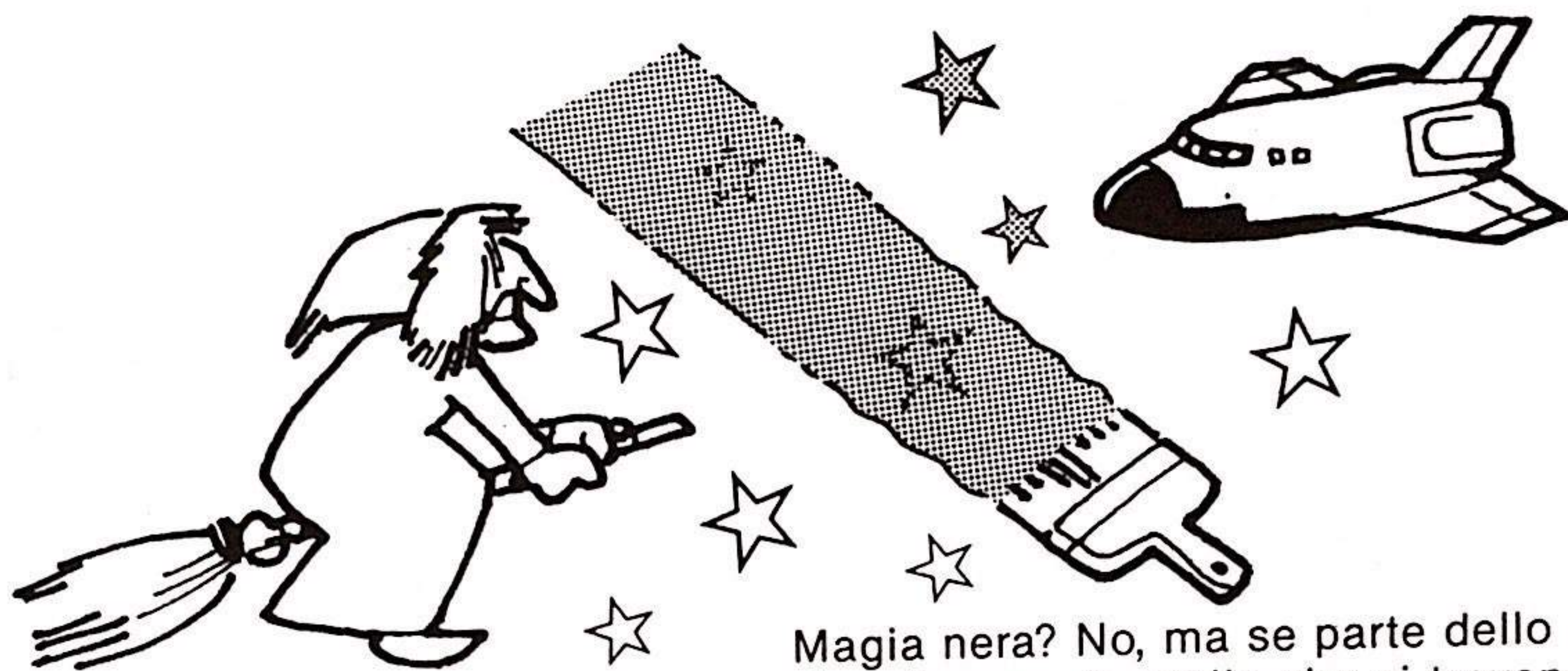


Siamo pronti: RUN.

Stelle nel cielo notturno!



Ma non abbiamo finito ancora. Possiamo far sembrare le nostre stelle delle stelle vere. Innanzitutto, controlliamo il numero delle stelle da far apparire—prima che il nostro schermo diventi una galassia. Usiamo **CTRL** e **STOP** come al solito.



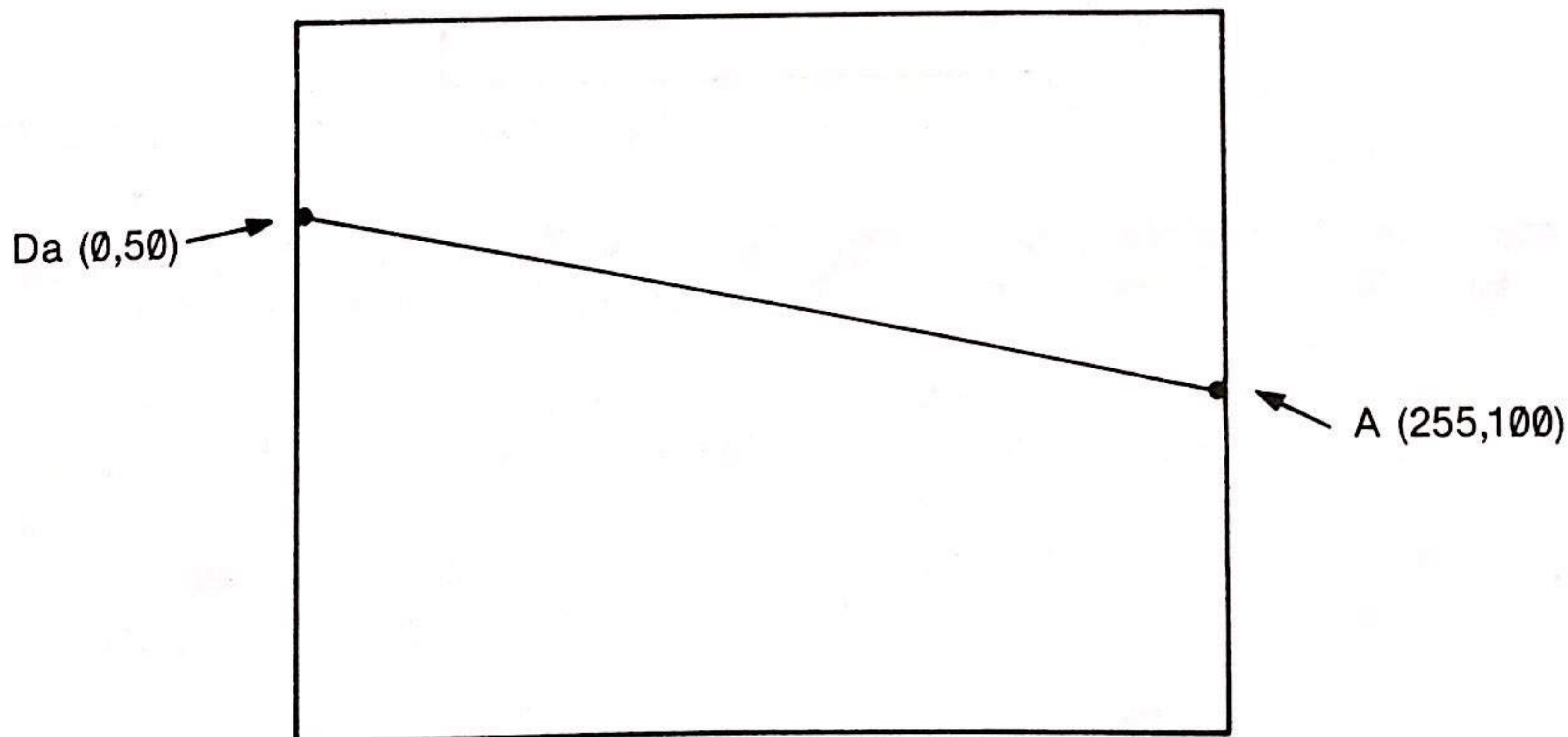
Magia nera? No, ma se parte dello schermo diventa nero, le stelle che si trovano in quella parte sono invisibili.

Se tracciamo delle **linee nere** sullo schermo, in un certo senso anneriamo una parte dello schermo. Il comando PSET serve solo per i punti: per tracciare delle linee useremo **LINE**. Questo è il comando che dice al computer di tracciare una linea e quale colore usare.

`LINE (0,50) - (255,100),2`

Per il trattino [-] nel mezzo, usate il segno meno.

Il trattino significa "da/a" e infatti questo comando traccia una linea **dal-**la posizione (0,50) **alla** posizione (255,100). Sappiamo, naturalmente, che l'ultima parte di questo comando sta per il colore verde medio. Se usiamo questo comando LINE nel nostro programma, ci tratterà una riga verde come questa:

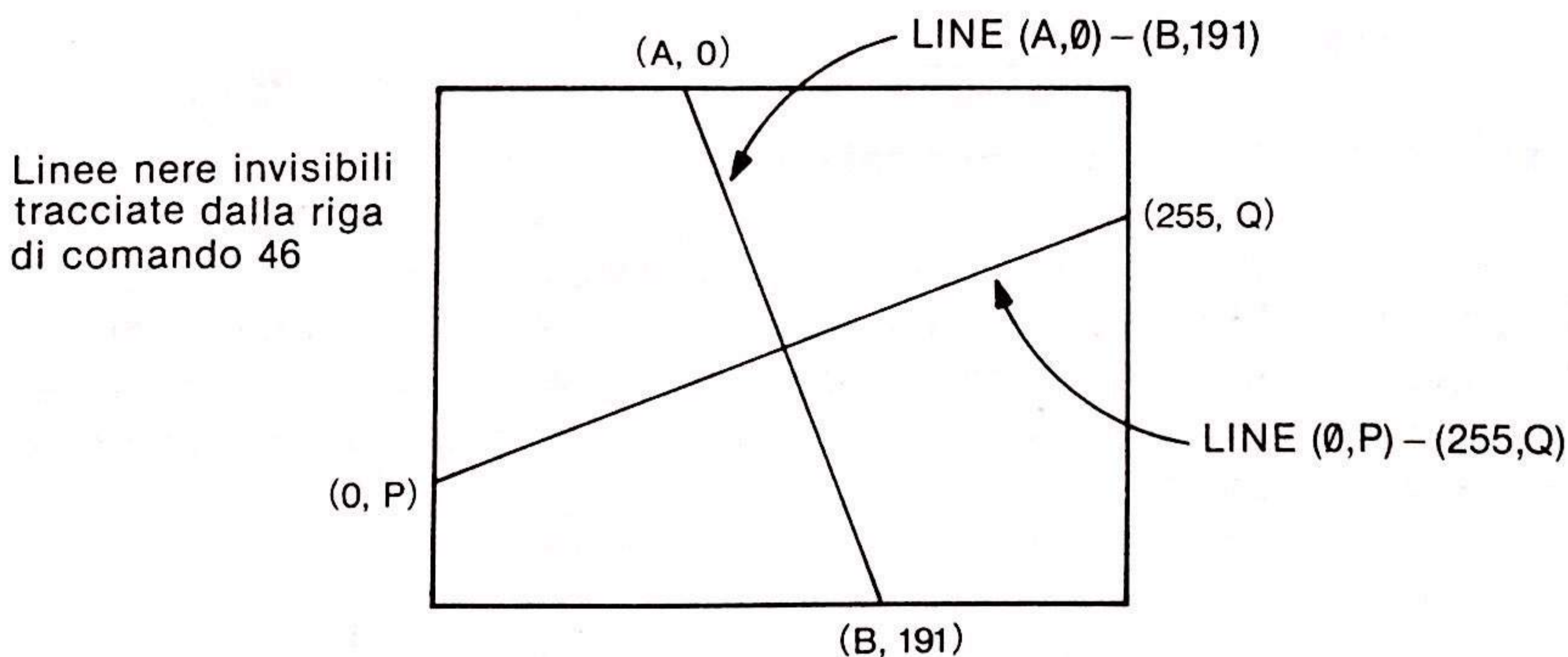


Ora che abbiamo capito il comando LINE proviamo ad usarlo per migliorare il programma delle stelle. Vogliamo usare una linea nera per rendere invisibili alcune stelle, no? Ciò significa che dovremo scegliere il codice di colore 1. Anche qui, inoltre, useremo delle variabili per far apparire una serie di linee casuali, insieme con una serie di punti casuali. Immettiamo nel computer queste tre righe:

```
42 A=INT(RND(1)*256):B=INT(RND(1)*256)
44 P=INT(RND(1)*192):Q=INT(RND(1)*192)
46 LINE (A,0)-(B,191),1:LINE (0,P)-(255,Q),1
```

Queste tre righe occupano più di una riga sullo schermo ma questo non è un problema. Il computer scrive automaticamente ogni comando su due righe, le quali costituiranno però sempre **una riga** perché c'è solo **un numero di riga** (42, 44 e 46).

La riga numero 46 contiene due comandi LINE che impiegano quattro nuove variabili. Le righe 42 e 44 sono dei comandi di numeri casuali, che già conosciamo, che assegnano dei valori alle nuove variabili A, B, P e Q. Un paio di linee casuali dovrebbero avere più o meno questo aspetto sullo schermo:



Dopo aver immesso le nuove righe di comando 42, 44 e 46, premiamo **RETURN** e poi battiamo LIST per vedere l'intero programma.


```

5 COLOR 15,1,1
10 SCREEN 2
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
35 C=INT(RND(1)*14)+2
40 PSET (X,Y),C
42 A=INT(RND(1)*256):B=INT(RND(1)*256
)
44 P=INT(RND(1)*192):Q=INT(RND(1)*192
)
46 LINE (A,0)-(B,191),1:LINE (0,P)-(2
55,Q),1
50 GOTO 20

```

Notate che le nuove righe di comando vengono prima del ciclo GOTO alla riga 50. Ciò significa che verranno ripetute all'infinito, proprio come le stelle. Ogni volta che il computer esegue i comandi compresi tra la riga 20 e la riga 46, fa apparire dei nuovi punti in posti sempre diversi e delle nuove linee in posti diversi. Quando un punto viene a trovarsi su una delle righe nere, diventa invisibile. E ora, diamo il comando RUN ... e le nostre stelle **brilleranno!**

Per poter vedere veramente queste linee e capire meglio come si comportano, cambiamo il numero di codice del colore nel comando 46 da 1 a 2. In tal modo vedremo delle righe verdi apparire sullo schermo.

NOTA: Se fermiamo il programma a questo punto lo schermo rimarrà nero. Possiamo riportarlo al colore blu scuro originario dando il seguente comando:

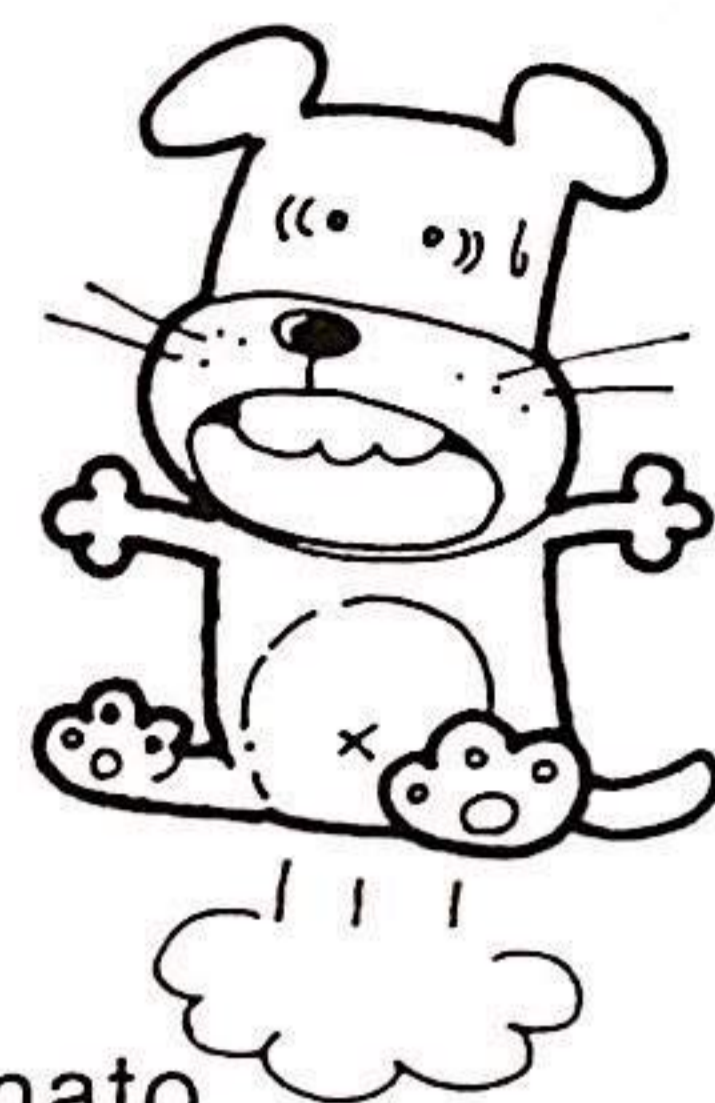
```
COLOR 15,4
```

15 sta per lettere bianche, 4 per sfondo e bordo blu scuro.

Non abbiamo ancora detto che attivando il BASIC all'inizio, lo schermo viene posto automaticamente nella condizione SCREEN 0. In questa condizione, il colore dei bordi è sempre uguale a quello dello sfondo. Se volete avere uno sfondo di colore diverso da quello dei bordi, provate il comando SCREEN 1 e quindi date un comando COLOR, il comando COLOR 15,4,7, per esempio.

Per tornare alla condizione precedente dello schermo, date il comando SCREEN 0.

■ CONSERVIAMO I NOSTRI PROGRAMMI



Come...

- Collegare il computer al registratore
- Trasferire un programma sul registratore
- Controllare se il programma è stato immagazzinato
- Caricare un programma dal registratore sul computer

Avete mai visto in fotografia un grande computer—uno di quelli che vengono usati per guidare un razzo spaziale, per controllare una fabbrica o per svolgere delle ricerche mediche? Forse vi è capitato di vedere delle parti di questi computer che assomigliano a dei grandi registratori. In effetti, è proprio quello che sono. Nei grandi computer queste parti sono chiamate “unità di pilotaggio di nastri magnetici” e servono a registrare e immagazzinare le informazioni usate dal computer.

Una parte di ciò che viene immagazzinato da queste unità sono i programmi—le istruzioni per la macchina come PSET, SCREEN, PRINT ed altri comandi che già abbiamo incontrato. Oltre ai programmi, queste macchine immagazzinano anche **dati**—cose come il punteggio di giochi, le soluzioni di problemi, formule scientifiche.

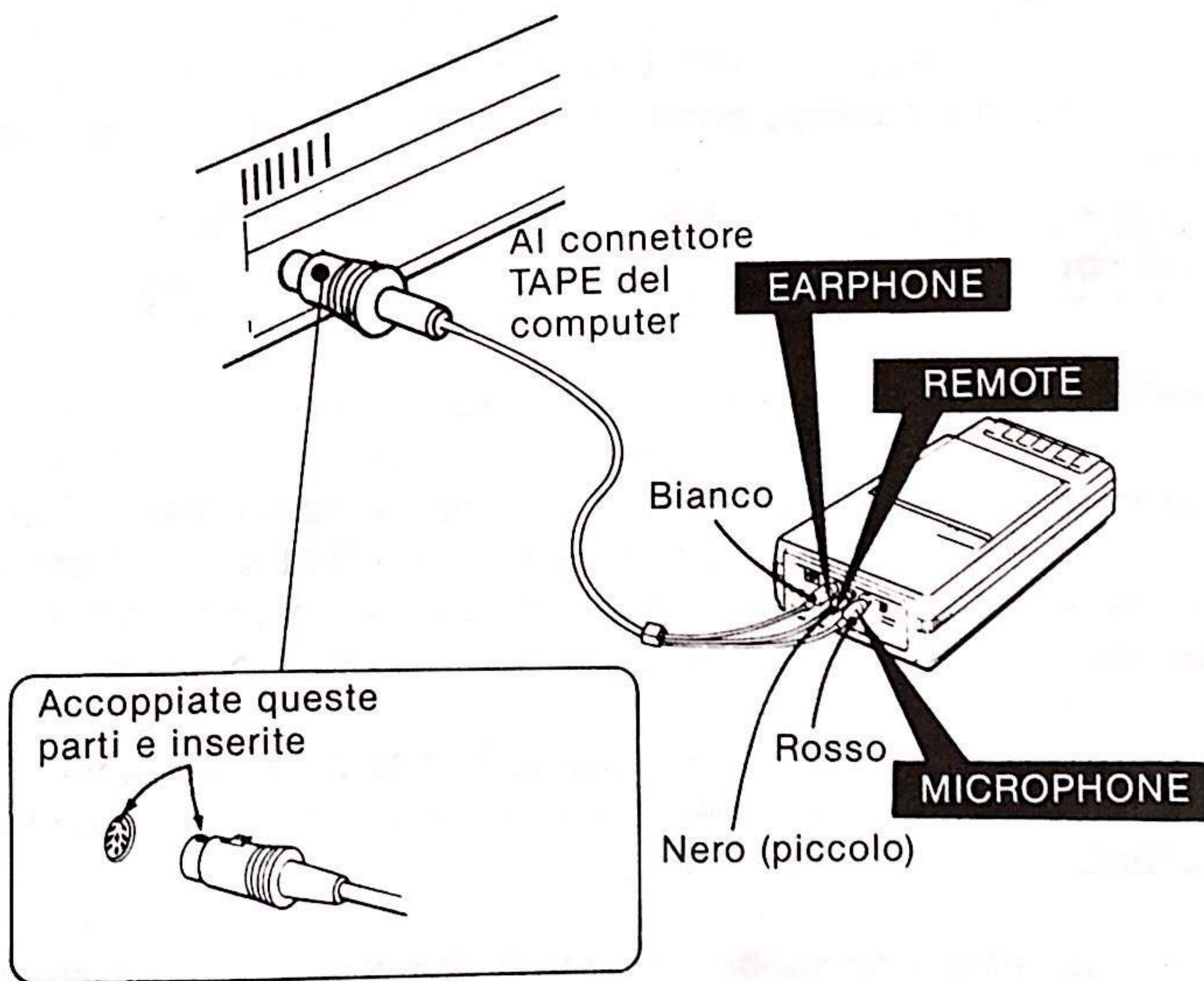
I computer si sono sviluppati molto rapidamente, al punto che operazioni che un tempo rendevano necessario l'impiego di un grande computer, di un'unità centrale di elaborazione, oggi possono essere eseguite da macchine di dimensioni ridotte come il nostro computer Sony. Il nostro computer è così piccolo che anche un bambino è in grado di sollevarlo, ma è più veloce e più sveglio di quei vecchi computer che costavano migliaia e migliaia di dollari e riempivano locali interi.

Non ci deve perciò stupire il fatto che anche il nostro computer Sony sia in grado di immagazzinare programmi e informazioni su nastri magnetici. Essendo più piccolo di un'unità centrale di elaborazione, usa un'unità di pilotaggio di nastri di dimensioni inferiori. Il nostro computer può usare un normale registratore a cassette, di quelli che si usano a casa per ascoltare musica o per studiare.

COLLEGAMENTO DEL REGISTRATORE

Il modo più semplice per collegare il computer al registratore è usare il cavo speciale che vi è stato fornito con il computer. Un capo del cavo termina con una spina rotonda di metallo con degli spinotti nel mezzo. Questo capo del cavo va inserito nella presa contrassegnata con TAPE situata sul retro della tastiera.

L'altro capo del cavo si divide in tre parti colorate di rosso, di bianco e di nero. La parte rossa va collegata alla presa con su scritto MICROPHONE del vostro registratore, mentre quella bianca va inserita nella presa contrassegnata EARPHONE. Se il vostro registratore è provvisto di una presa contrassegnata REMOTE, inseritevi l'estremità nera del cavo (se il vostro registratore non è fornito della presa REMOTE, lasciate semplicemente scollegata l'estremità nera).



A questo punto il computer e il registratore sono pronti per parlare l'uno con l'altro.

FACCIAMO PARLARE IL COMPUTER

Perché il computer comunichi al registratore il contenuto della sua memoria dobbiamo dargli il seguente comando:

```
CSAVE "nome di file"
```

La C di CSAVE significa cassetta, mentre SAVE vuol dire: conservare il programma contenuto nella memoria del computer registrandolo sulla cassetta. "Nome di file" (attenzione a non dimenticare le virgolette!) può essere un qualsiasi nome che noi vogliamo dare al nostro programma, per poterlo—noi, il registratore e il computer—distinguere dagli altri.

Un **nome di file** può essere composto da un massimo di sei caratteri. I caratteri possono essere lettere, numeri o segni grafici ma il **primo** carattere deve essere una **lettera** dell'alfabeto.

STAR, ad esempio, è un buon nome di file per il programma che abbiamo eseguito nel capitolo precedente. È composto da quattro caratteri, inizia con una lettera ed è comodo perché ci ricorda il contenuto del programma, le stelle.

Conserviamolo:

```
CSAVE "STAR"
```

Aspettiamo però a premere **RETURN**. Se premiamo **RETURN**, il computer comincerà immediatamente a inviare al registratore le informazioni da conservare. Dobbiamo perciò essere sicuri che il registratore sia pronto a riceverle.

Se il registratore è provvisto di una presa REMOTE nella quale abbiamo già inserito il cavo di collegamento, mettiamo il registratore in modo di registrazione.

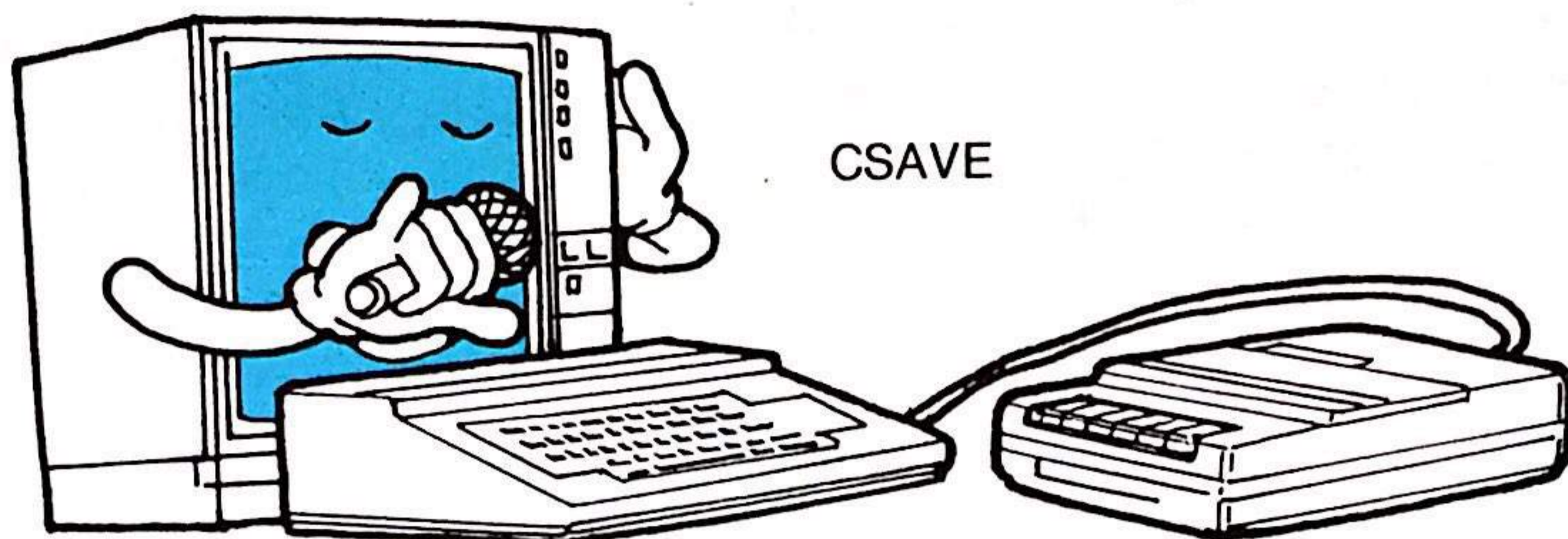
Come vediamo, anche premendo il tasto di registrazione il nastro non si muove. Il computer, infatti, lo tiene "sotto controllo". Quando premiamo **RETURN**, il nastro comincia a girare, registra il programma e poi si spegne da solo. Alla fine della registrazione sullo schermo apparirà

```
CSAVE "STAR"
```

```
OK
```



Se il registratore non è provvisto di una presa REMOTE, premiamo il tasto **RECORD**. Così facendo, il nastro comincerà immediatamente a girare. **Aspettiamo** alcuni secondi che la velocità del nastro si stabilizzi e poi premiamo **RETURN**. Quando sullo schermo appare l'indicazione Ok, premiamo il tasto **STOP** del registratore e avremo quasi finito.



Il programma si trova in due posti. È ancora contenuto nella memoria del computer e, se tutto è andato bene, è anche inciso sul nastro magnetico.

SIAMO SICURI CHE IL NASTRO HA REGISTRATO IL PROGRAMMA? _____

Bisogna **sempre controllare** che il **registrator**e abbia effettivamente registrato il nostro programma—prima di fare qualsiasi altra cosa. L'operazione di controllo viene eseguita dal computer stesso che confronterà la sua memoria con la registrazione. Prima di tutto, **scolleghiamo** il cavo dalla presa "REMOTE". Poi, **riavvolgiamo** il nastro fino al punto in cui si trovava prima che il programma fosse registrato. **Regoliamo il volume** del registratore su 1/2 circa o sul massimo. Quindi, **ricolleghiamo** la spina a REMOTE e battiamo questo comando:

```
CLOAD? "STAR"
```

Se il registratore non è fornito di presa "REMOTE", battiamo **RETURN** prima di porre il registratore in modo di riproduzione, cioè prima di far girare di nuovo il nastro. Se il registratore ha la presa "REMOTE", mettiamolo in modo di riproduzione. Il nastro comincerà a girare automaticamente quando premeremo **RETURN**. Quando il computer sente l'inizio del programma farà apparire sullo schermo:

```
Found:STAR
```


Riascoltando il nastro il computer confronterà punto per punto il programma registrato con quello contenuto nella sua memoria e, se tutto è corretto, farà apparire Ok sullo schermo. (Attenzione a non dimenticare di fermare manualmente il nastro se il registratore fosse sprovvisto di telecomando).

Questa operazione è molto importante—se non vogliamo che tutta la fatica fatta per scrivere il programma vada sprecata—e talvolta può presentare dei problemi. Se il messaggio "Found" (trovato) non appare sullo schermo, può darsi che non abbiamo riavvolto abbastanza il nastro.

Se il segno Ok **non** appare sullo schermo, aumentiamo il volume del registratore e riproviamo. Se nemmeno questo risolve il problema, significa che c'è un'interferenza elettronica oppure che il programma non è stato registrato correttamente. In tal caso bisognerà controllare i collegamenti e ricominciare da CSAVE "STAR".

Se invece durante l'operazione di controllo non ci sono problemi e sullo schermo appare Ok, sappiamo che il programma è stato registrato sul nastro. Possiamo perciò etichettare quel nastro, riporlo e usarlo quando ne avremo bisogno in futuro.

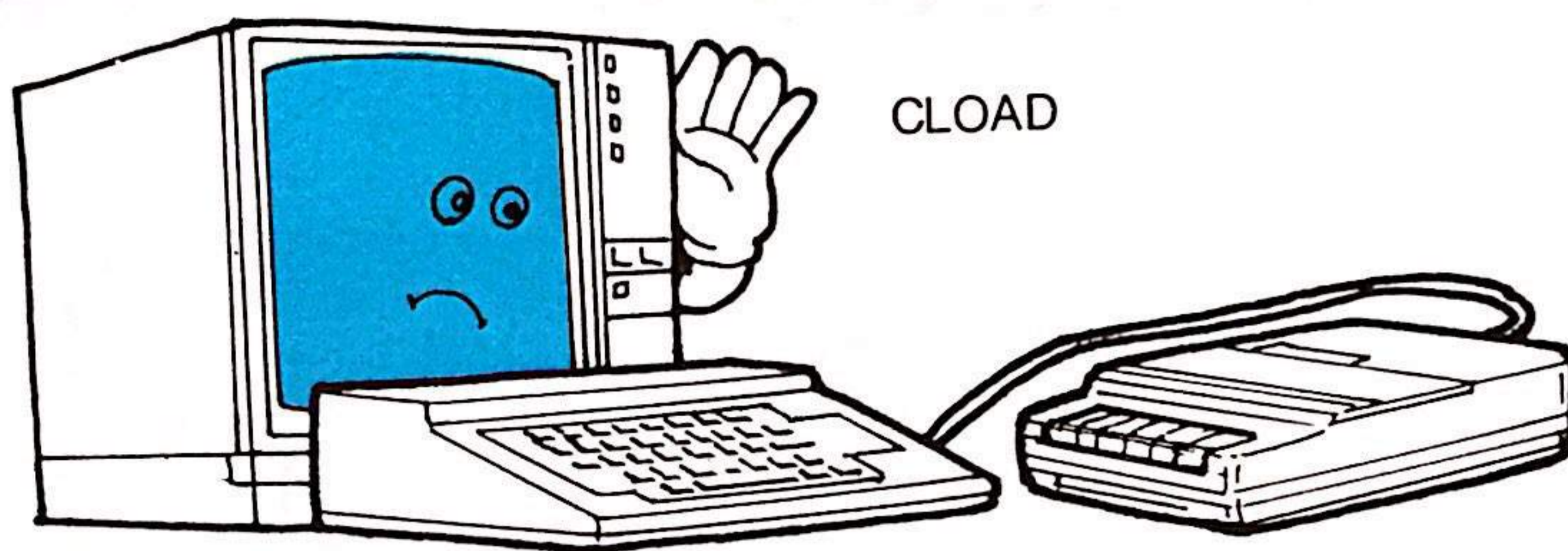
Quando siamo **sicuri** che il programma è stato registrato sano e salvo sul nastro, possiamo immettere nel computer NEW o battere il tasto **RESET**. In questo modo si cancella la **memoria del computer**—ma non la **memoria esterna** al sicuro sul nastro.

CARICHIAMO UN PROGRAMMA _____

"Caricare il programma" nel computer significa usare di nuovo quel nastro. Questo è il momento buono per fare un po' di esercizio. Per caricare un programma si procede quasi nello stesso modo dell'operazione precedente. Assicuriamoci che il registratore sia disposto in modo da **parlare** dentro il computer e che il nastro sia al posto giusto—esattamente prima dell'inizio del programma registrato. Diamo poi questo comando:

```
LOAD "STAR"
```

Notate che stavolta non c'è il punto di domanda (?). È ovvio che se usiamo un nome di file diverso dobbiamo batterlo all'interno delle virgolette al posto di STAR.



Quando il computer avrà finito di ascoltare il nastro, sullo schermo apparirà

```
CLOAD "STAR"  
Found:STAR  
OK  
■
```

Assicuriamoci che il nastro si sia fermato—ed eccoci pronti a battere RUN per far girare il nostro programma nel computer.

Talvolta sullo schermo potrebbe apparire

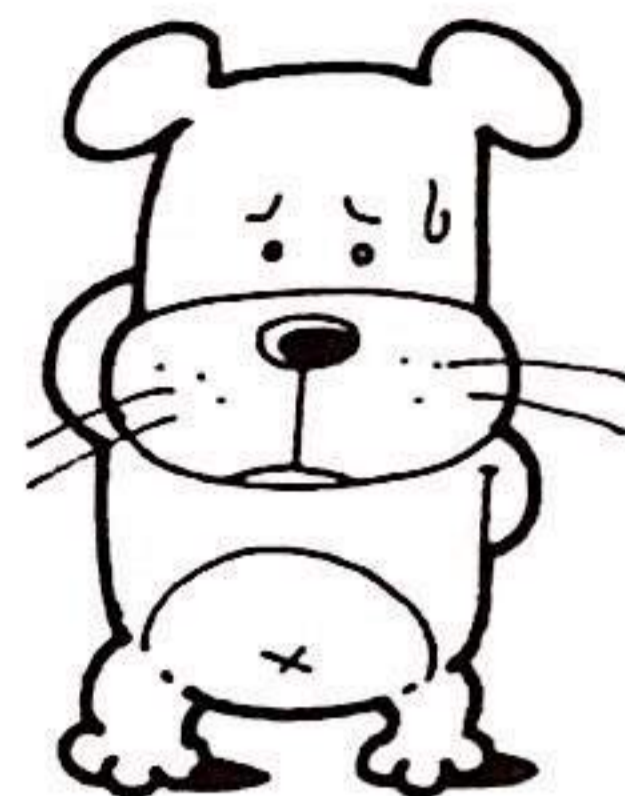
```
Device I/O error
```

I/O significa **ingresso/uscita** e si riferisce in questo caso al collegamento col registratore che non è perfetto (error). In questo caso, regoliamo un po' il volume e riproviamo a caricare il nastro.

DECISIONI E GIOCHI

Come...

- Usare il computer per prendere delle decisioni
- Programmare una formula condizionale
- Fare un gioco sul nostro computer
- Semplificare il nostro programma
- Migliorare il nostro programma



Vi ricordate Fido, il nostro buon cane di famiglia? È molto bravo ad eseguire ordini semplici, come "Seduto", "Portami" e "Da' la zampa". Finora, anche il nostro Computer Sony ha eseguito dei compiti facili, non molto diversi da quelli di Fido—come "PRINT", "GOTO", "PSET", e così via.

Poi abbiamo incontrato Supercane che ha eseguito dei compiti piuttosto difficili basati su "se la palla è nera..." o "se la palla è bianca...". Supercane era capace di **prendere decisioni**. Sappiamo già che il nostro Computer Sony è intelligente abbastanza per fare queste cose, perciò cominceremo a scrivere dei "super-programmi" in grado di prendere delle decisioni.

Per cominciare, cancelliamo con il comando NEW l'ultimo programma e quindi immettiamo:

```
10 A=INT(RND(1)*5)+1
20 INPUT B
30 IF A=B THEN GOTO 50
40 GOTO 10
50 PRINT "Azzecato!"
60 GOTO 10
```

Dopo averlo battuto correttamente, diamo il comando RUN e poi battiamo **RETURN**.

```
RUN
? ■
```


Non è la prima volta che il computer ci risponde con un punto interrogativo. Stavolta però il cursore si trova sulla stessa riga e non su quella successiva. Questo significa che il computer sta aspettando che noi **immettiamo** qualcosa—perché c'è un comando INPUT alla riga 20. Come sapete, per immettere qualcosa—in questo programma ogni numero dall'1 al 5—basta premere uno dei tasti. Diamo al computer il numero 3 (cosa succederà se immettiamo una lettera o un grafico?).

Premiamo `3` e `RETURN`. Cosa appare sullo schermo ora?

```
? 3
Azzecato!
? ■
```

```
? 3
? ■
```

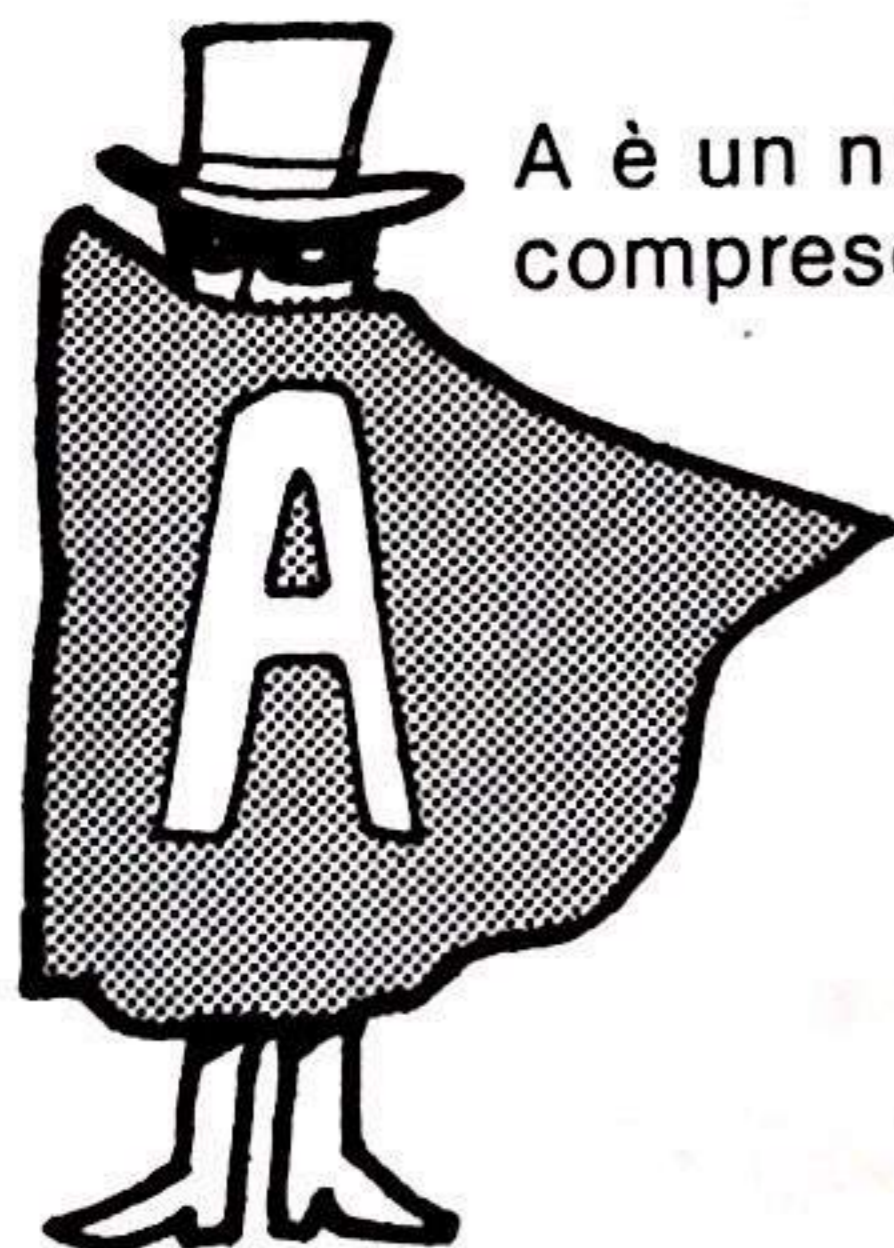
Entrambi gli schermi ci danno un punto di domanda alla fine del programma, così sappiamo che il computer vuole che immettiamo un altro numero. E poi un altro. E un altro. E un altro ancora. Questo è il gioco: con il nostro numero abbiamo indovinato giusto—“Azzecato!”—o abbiamo sbagliato? Ogni quante volte riusciamo a ottenere un “Azzecato!”? (Secondo la legge delle probabilità dovremmo indovinare un numero ogni cinque tentativi, o due ogni dieci, o circa 20 ogni 100). Non c'è pericolo che il computer si stanchi di questo gioco prima di noi (non si stancherà mai), perciò premiamo `CTRL` e `STOP`.

Cosa succede qui? Diamo un'occhiata al programma. Alla prima riga c'è un nostro vecchio amico, il **comando di numero casuale**. Ogni volta che viene rimandato indietro attraverso il ciclo (riga 60), il programma sceglie un numero compreso tra 1 e 5 e lo assegna alla variabile A.

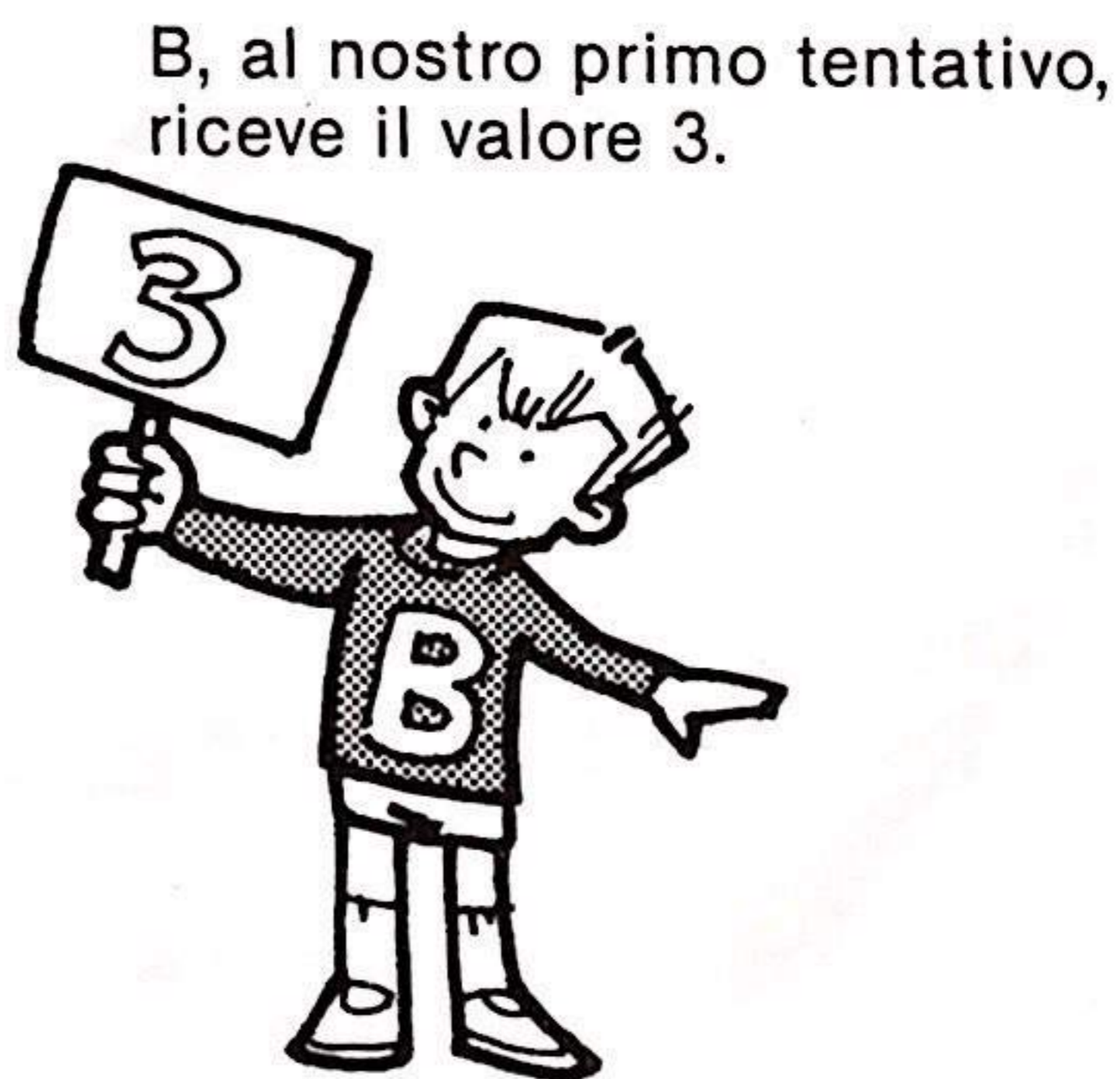
Alla riga 20 facciamo ora una nuova conoscenza:

```
20 INPUT B
```

B è una seconda variabile e sta aspettando che le assegnamo un valore dalla tastiera. Dopo che abbiamo battuto un numero sulla tastiera e che a entrambe le variabili è stato assegnato un valore, il computer passa a leggere la riga successiva.



A è un numero
compreso tra 1 e 5



B, al nostro primo tentativo,
riceve il valore 3.

AZZECCATO O SBAGLIATO—DECIDE IL COMPUTER

La riga 30 ci ricorda Supercane perché comincia con IF.

```
30 IF A=B THEN GOTO 50
```

La parola IF è **sempre** combinata con la parola THEN. Per esempio, “SE (IF) hai fame, ALLORA (THEN) devi mangiare”, oppure “SE hai un Computer Sony, ALLORA ti diverti”.

IF [condizione] THEN [qualcosa]

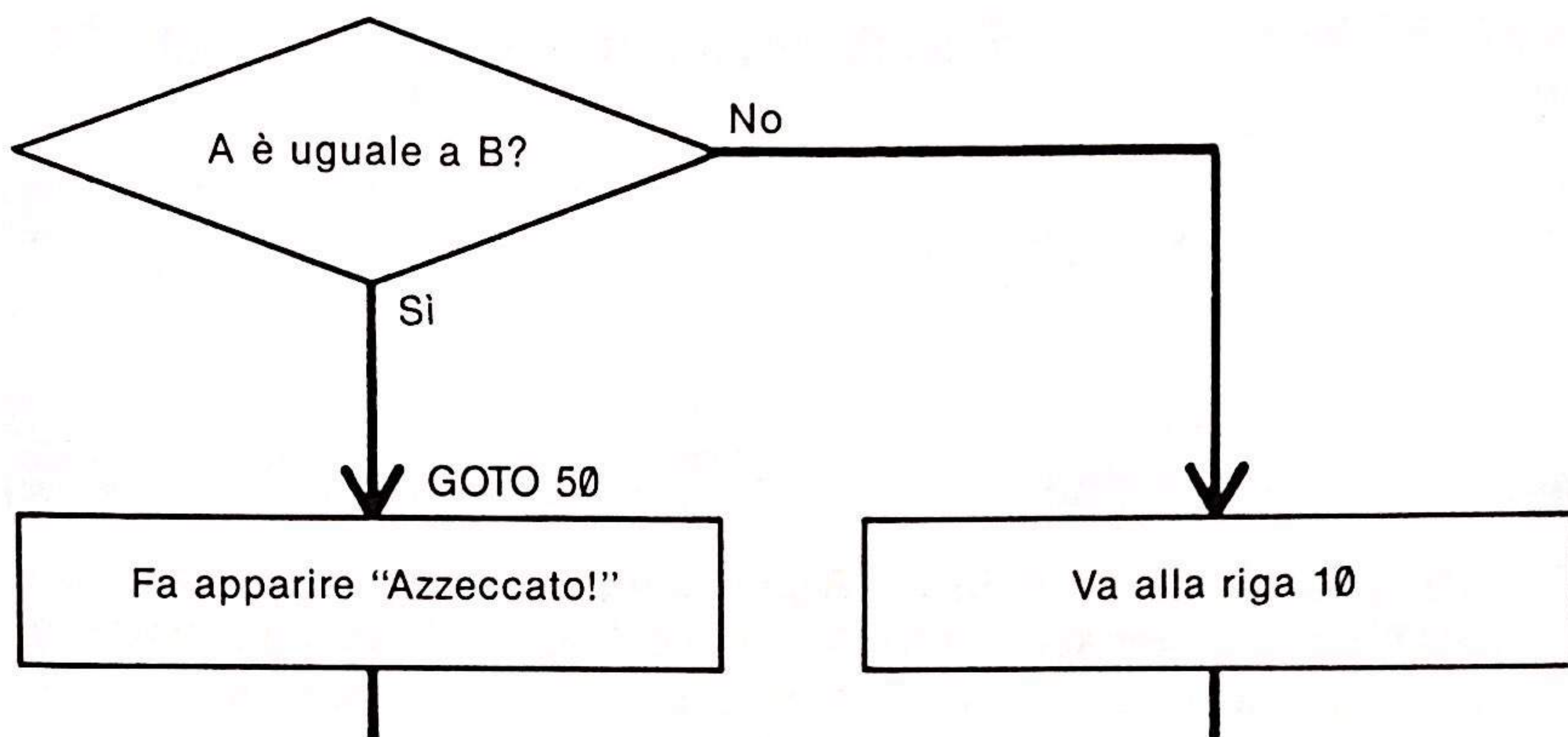
Quale può essere la **condizione** delle variabili A e B? Esse possono essere uguali (A=B), A può essere maggiore (A>B), oppure B può essere maggiore (A<B).

“Qualcosa” può essere un comando qualsiasi. In questo caso è GOTO 50.

Questa viene chiamata **formula condizionale**. Ce ne sono sei nel BASIC ed alcune possono essere scritte in due modi diversi.

Simbolo	Significato	Esempio	
=	Uguale a	IF A=B	Se A è uguale a B
>	Maggiore di	IF A>B	Se A è maggiore di B
<	Minore di	IF A<B	Se A è minore di B
>=	} Uguale a o } maggiore di	IF A>=B	Se A è uguale o maggiore di B
=>		IF A=>B	
<=	} Uguale a o } minore di	IF A<=B	Se A è uguale o minore di B
=<		IF A=<B	
<>	} Diverso da	IF A<>B	Se A è diverso da B
><		IF A><B	

La riga 30 è quindi una formula condizionale. Se la parte che segue IF è vera (A=B), il computer va al comando indicato (GOTO 50). Ma se A e B non sono uguali, il computer passa alla riga successiva del programma, la riga 40, senza prendere in considerazione il comando THEN. La riga 40, naturalmente, rimanda il programma all'inizio a generare un nuovo numero casuale—che noi cercheremo di indovinare.



È chiaro ora che “Azzecato!” può apparire solo quando A e B—e B è il numero che noi forniamo tirando a indovinare—sono uguali. “Azzecato!” significa che il numero che noi abbiamo immesso come valore della variabile B è uguale a quello che il computer ha assegnato alla variabile A.

Dopo aver fatto apparire “Azzecato!”, il computer si porta alla riga 60 e il gioco ricomincia. Il gioco continua in ogni caso, sia che A e B siano uguali o che non lo siano.

SEMPLIFICHIAMO IL PROGRAMMA

I programmi per computer andrebbero sempre scritti il più semplicemente possibile di modo che siano facili da capire da un lato e offrano meno possibilità di errori di battitura dall'altro. Diamo un'altra occhiata al programma dell'indovina un numero.

```

10 A=INT(RND(1)*5)+1
20 INPUT B
30 IF A=B THEN GOTO 50
40 GOTO 10
50 PRINT "Azzecato!"
60 GOTO 10

```

Sappiamo che se $A = B$ il computer farà apparire un “Azzecato!”. Una sola azione andrebbe meglio espressa da un solo comando—perciò cambiamo la riga 30 a

```
IF A=B THEN PRINT "Azzecato!"
```


In questo modo non abbiamo più bisogno dei comandi delle righe 50 e 60, no?

```
10 A=INT(RND(1)*5)+1
20 INPUT B
30 IF A=B THEN PRINT "Azzecato!"
40 GOTO 10
```

Così è molto più semplice! Se A e B sono uguali, dopo aver stampato "Azzecato!" il computer scenderà alla riga 40 e quindi, leggendo GOTO 10, tornerà alla riga 10. Se A e B non sono uguali, il computer farà la stessa cosa, ma senza stampare "Azzecato!". Il nostro gioco ora funziona con soli quattro comandi.

E ora facciamo queste modifiche al programma che appare sullo schermo. Prima di tutto, cambiamo il comando della riga 30 e scriviamo PRINT "Azzecato!" al posto di GOTO 10. Quindi, cancelliamo le righe 50 e 60 in questo modo: portiamo il cursore sotto la riga 60 e immettiamo 50. Premiamo **RETURN** e in tal modo la riga 50 viene cancellata dalla memoria del computer. Facciamo lo stesso con il 60 per cancellare la riga 60.

Le righe 50 e 60 sono sparite dallo schermo? No, sono ancora lì. Ma sono state cancellate dalla memoria del computer. Se diamo il comando LIST, il computer ci mostrerà quello che è ora il contenuto della sua memoria:

```
LIST
10 A=INT(RND(1)*5)+1
20 INPUT B
30 IF A=B THEN PRINT "Azzecato!"
40 GOTO 10
```

Così è meglio, no?

Alcune sottigliezze

Ora che abbiamo reso il programma dell'indovina un numero il più semplice possibile, possiamo pensare a come renderlo migliore, aggiungendovi cose utili e divertenti allo stesso tempo. Man mano che le aggiungiamo, naturalmente, cercheremo di mantenere il più semplice possibile ogni miglioramento.

Quando A è uguale a B il computer ce lo dice facendo apparire il messaggio "Azzeccato!". Quando A e B sono diverse, il computer non ci dà nessun messaggio prima di riprendere il gioco. Il programma diventerà più facile da capire per gli amici che giocano con noi se cambieremo in questo modo la riga 30:

```
30 IF A=B THEN PRINT "Azzeccato!" ELSE PRINT  
"Sbagliato"
```

ELSE significa "altrimenti". È una parola molto utile se viene posta dopo una formula condizionale, perché rende più chiaro il punto soggetto a condizione all'interno del programma.

```
IF formula condizionale THEN comando 1 ELSE comando 2
```

Siccome questo comando è lungo più di 37 caratteri, "andrà a capo" sulla riga seguente—facendo momentaneamente sparire la riga 40. Ma niente paura—la riga è sempre nella memoria del computer: provate a usare il comando LIST per farla tornare sullo schermo! C'è un'altra cosa che possiamo fare per rendere questo programma più facile da capire per i nostri amici quando lo vedranno. Possiamo cambiare la riga 20 in questo modo:

```
20 INPUT "Prova a indovinare(1-5) ";B
```

Per capire meglio il significato di questo nuovo comando immettiamolo sullo schermo e diamo poi il comando RUN.

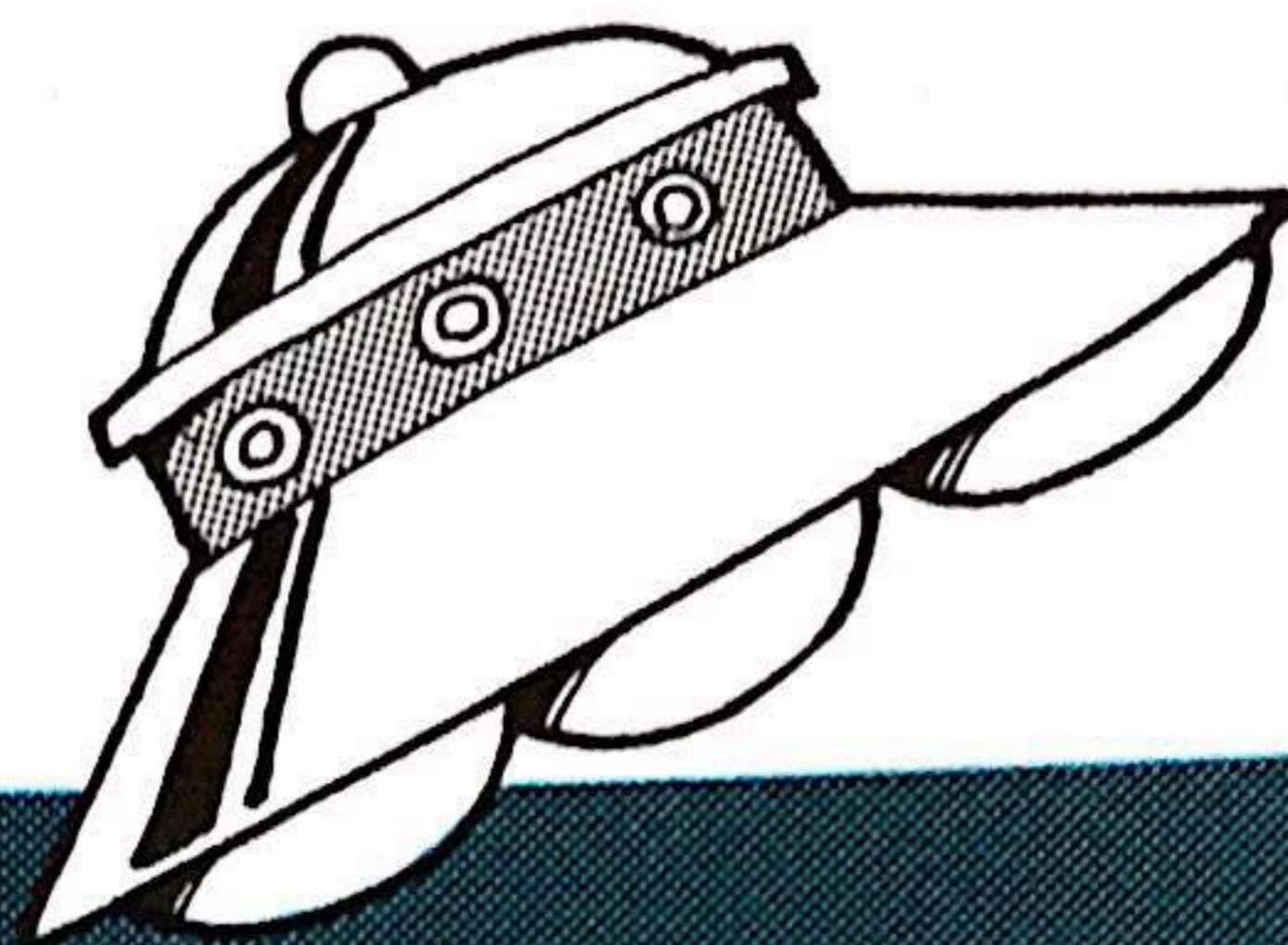
```
RUN  
Prova a indovinare(1-5) ? ■
```

Facile da capire, no? In questo modo è molto meglio del messaggio "?" che vedevamo sullo schermo pochi minuti fa. Ora potete fare questo gioco dell'indovina un numero insieme con i vostri amichetti e non ci vorrà molto tempo perché tutti capiscano il vostro programma.

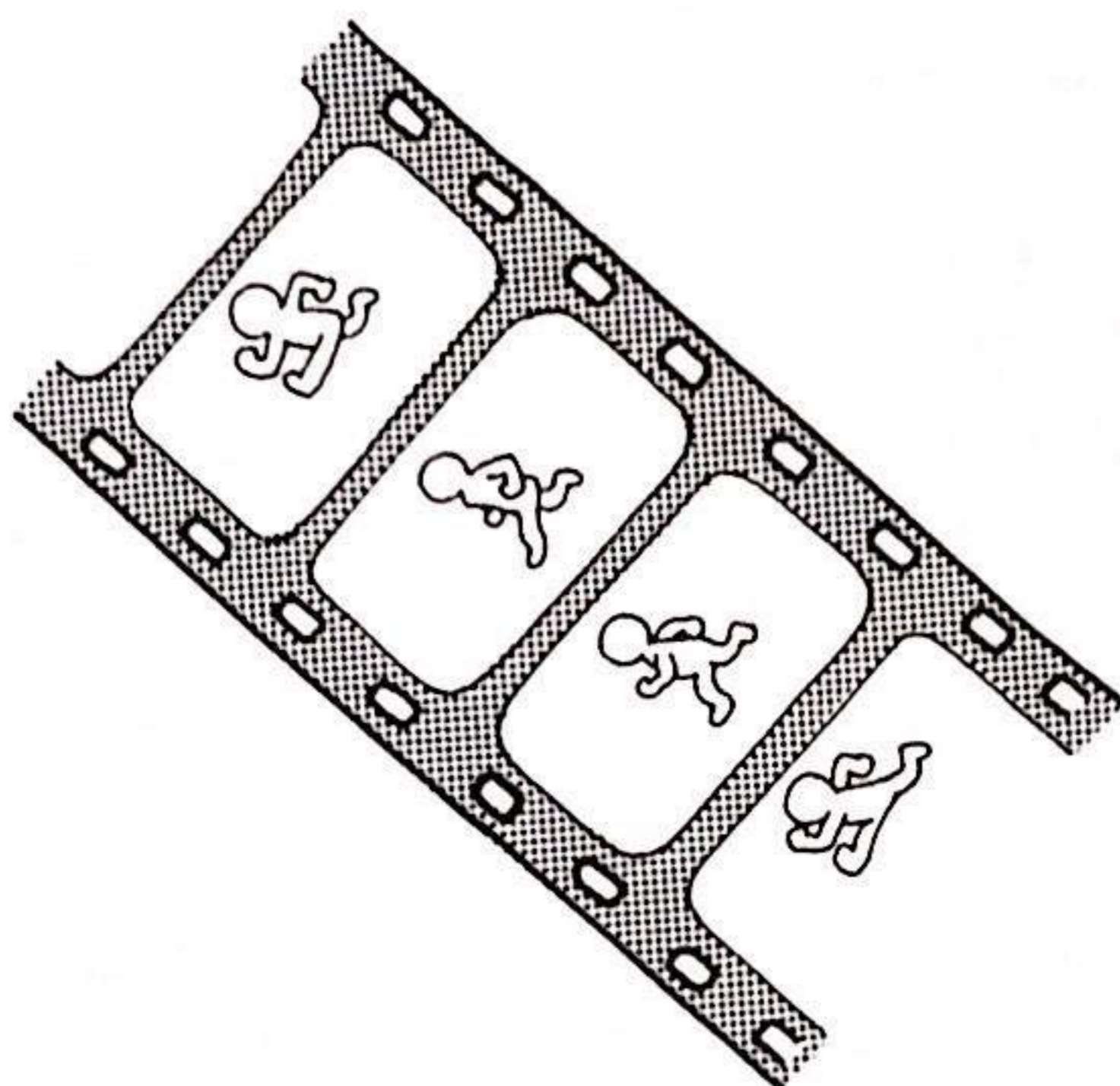
GRAFICI CHE SI MUOVONO

Come...

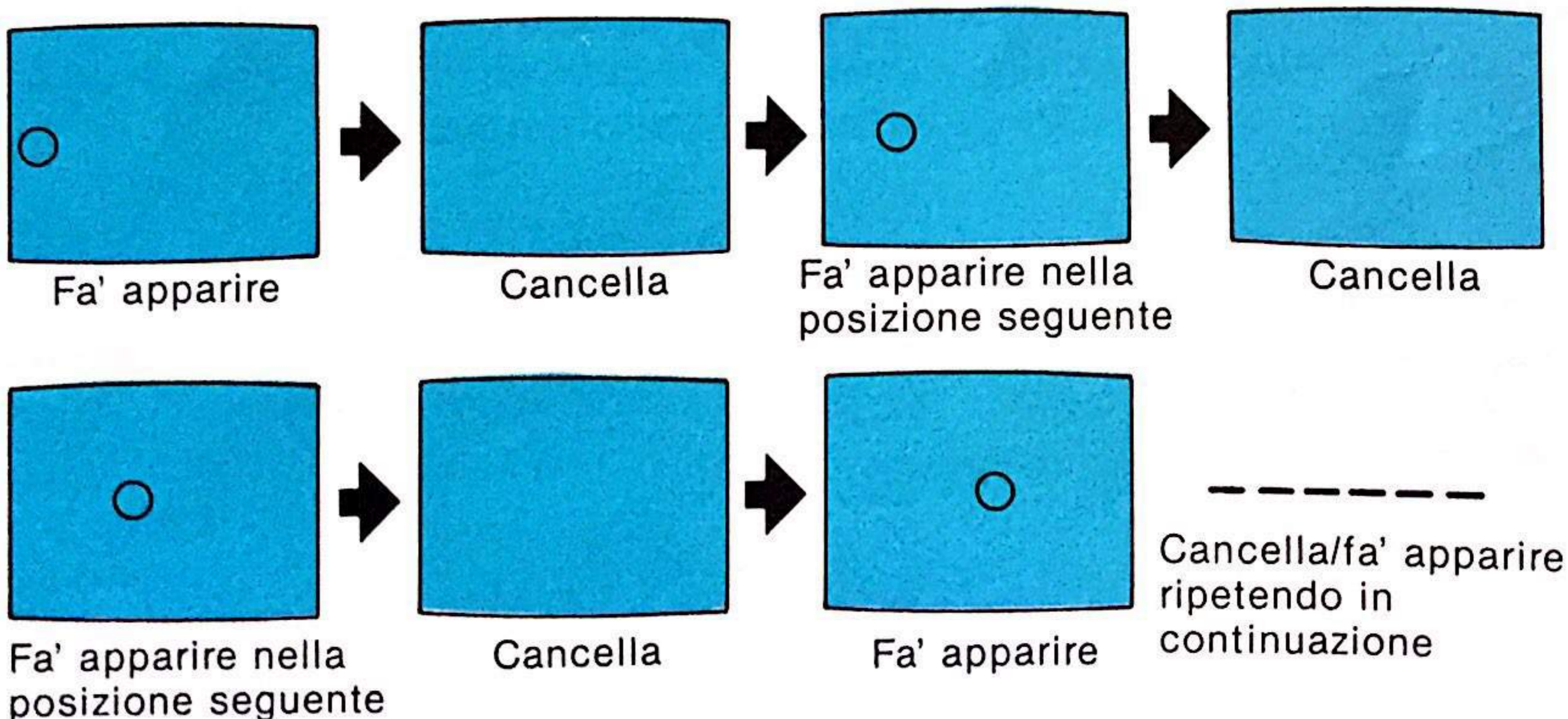
- Muovere i caratteri sullo schermo
- Fare un film sugli UFO
- Usare le variabili per accorciare il programma



Tutti sanno che le immagini di un film non si muovono realmente. Un film non è altro che una serie di immagini ferme che si susseguono rapidamente l'una dopo l'altra. Siccome ogni immagine è un po' diversa dall'altra, riceviamo l'illusione del movimento.



La televisione funziona nello stesso modo. L'immagine che noi vediamo sul nostro schermo televisivo cambia molte volte in un solo secondo, così noi abbiamo l'impressione che si muova. Lo schermo del nostro computer Sony è quasi uguale a quello di un televisore e può essere perciò usato per fabbricare delle immagini che si muovono. Si fa in questo modo:



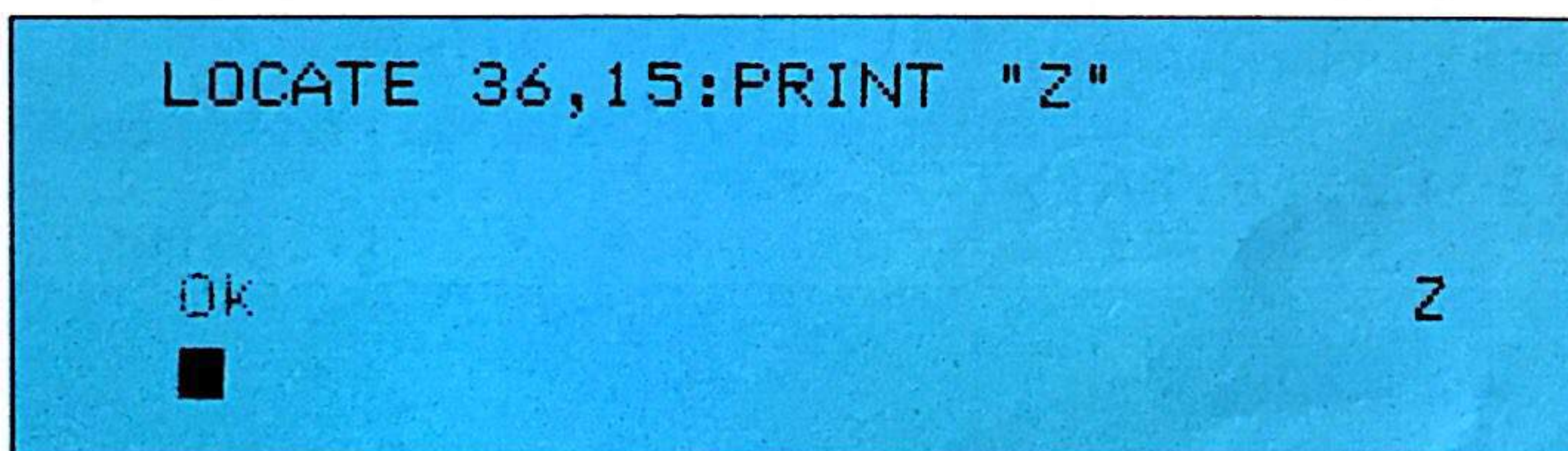
Per muovere il segno O attraverso lo schermo, lo facciamo apparire sulla sinistra, poi lo cancelliamo e lo facciamo apparire un po' più in là, poi lo cancelliamo ... e così via. Per fare ogni passo di questa serie, dobbiamo dire al computer di cancellare, di individuare poi la corretta posizione, quindi, di stampare qualcosa.

```
CLS  
LOCATE 36,15:PRINT "Z"
```

Ci sono due nuovi comandi, seguiti da PRINT. CLS significa "Clear screen" o "Schermo pulito". Se viene immesso da solo, questo comando fa sparire ogni altra cosa dallo schermo.

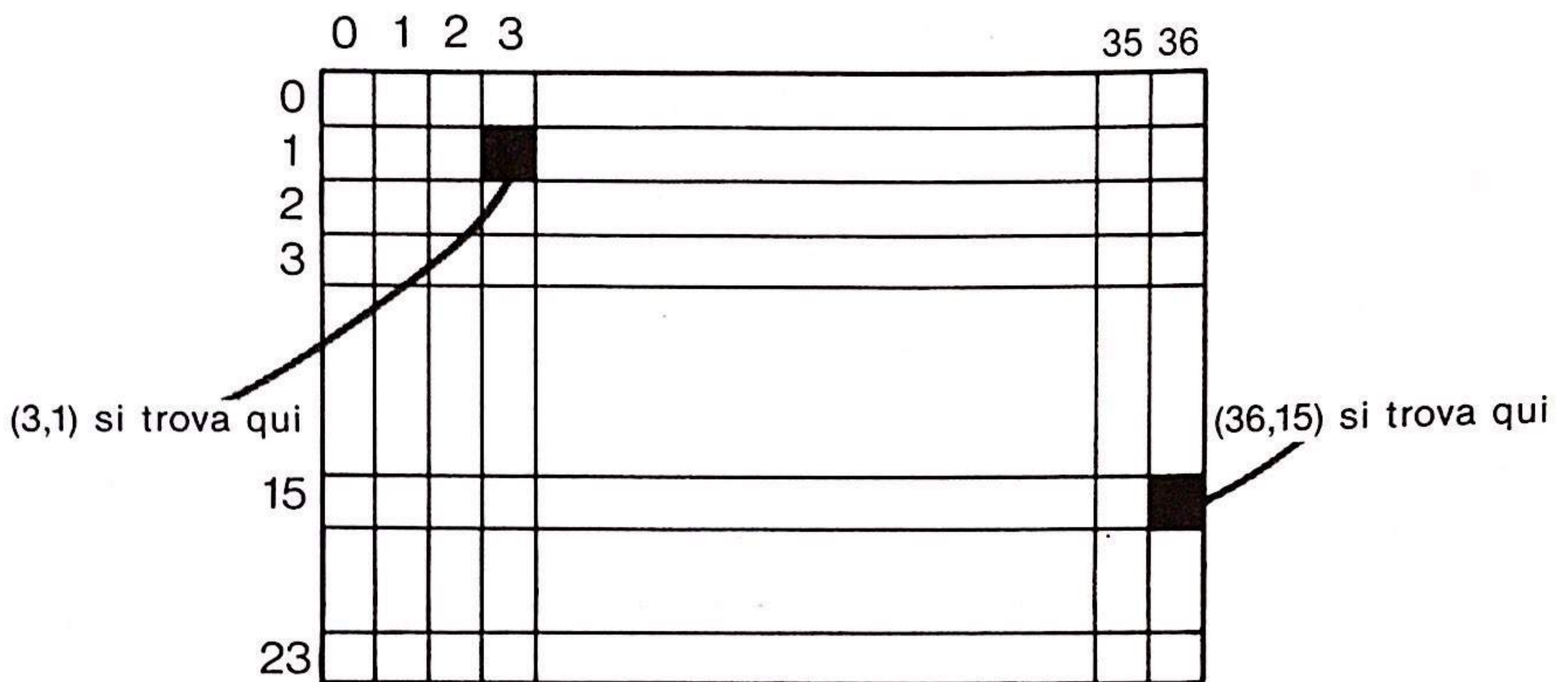


Per capire che cosa significa il comando LOCATE, immettiamo tutti insieme nel computer i tre comandi scritti sopra.



Il comando LOCATE 36, 15 dice al cursore di saltare 36 spazi e di scendere quindi alla riga 15. I **due punti** (:) dicono al computer che c'è un altro comando sulla stessa riga. Infine, il comando PRINT gli dice di far apparire la lettera Z nel punto in cui si trova il cursore.

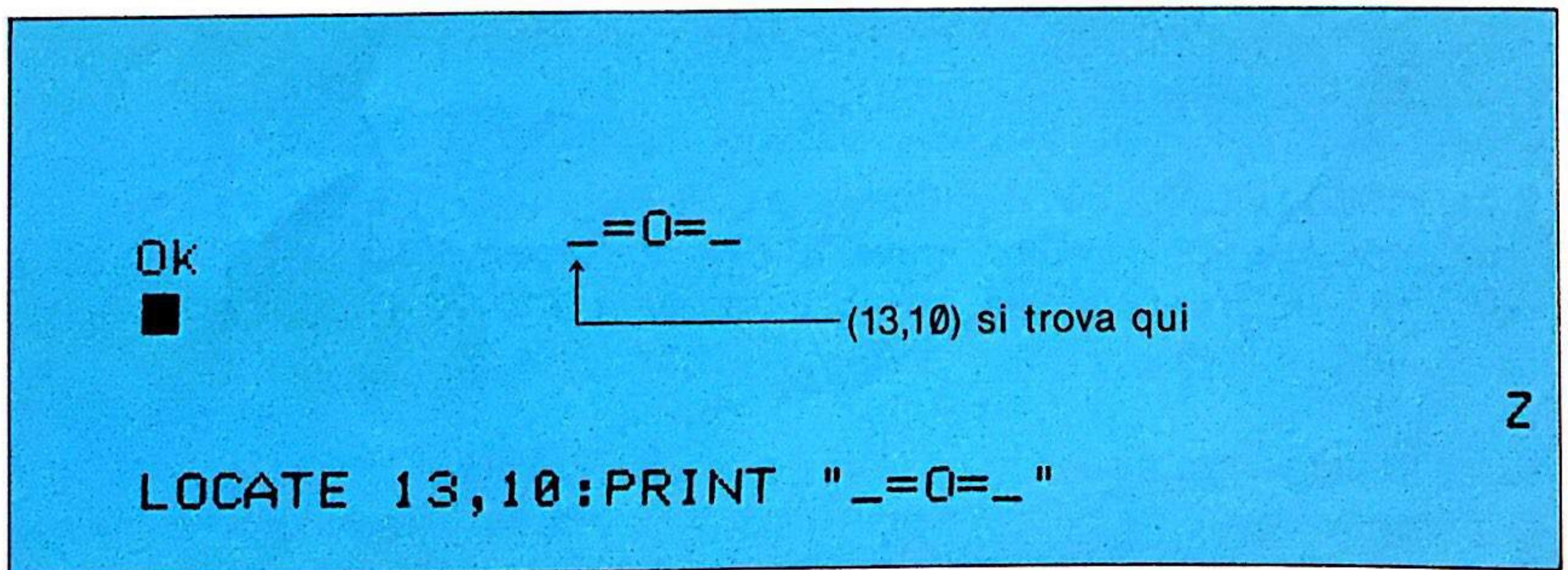
Questo comando assomiglia molto al comando PSET che abbiamo usato precedentemente per far apparire delle stelle luminose sullo schermo, con una differenza però: il carattere Z è più grande di un punto ed è un carattere, mentre un punto è un segno grafico. Per posizionare sullo schermo dei caratteri, quindi, dobbiamo dire LOCATE 36,15: PRINT "qualcosa"; per far apparire dei grafici, invece, diciamo per esempio PSET (36,18): il risultato è identico. In entrambi i casi possiamo considerare 36,15 una specie di **indirizzo**.



Il comando PRINT che segue il comando LOCATE e i due punti (:) non si limita a stampare caratteri singoli come "Z". Il comando LOCATE può posizionare sullo schermo **una qualsiasi quantità** di lettere o di parole. Il comando LOCATE non fa altro che dire al cursore dove andare prima di iniziare a eseguire PRINT. Si può usare LOCATE per iniziare a scrivere un paragrafo nel mezzo della pagina, per iniziare un gioco in fondo allo schermo o per posizionare tutto quello che si vuole.

Usiamo un po' la fantasia. Siete pronti a vedere un UFO? (gli UFO sono, naturalmente, degli "oggetti volanti non identificati"—navi spaziali da un altro pianeta—che probabilmente usano computer). Possiamo costruire un UFO con soli cinque caratteri:

```
LOCATE 13,10:PRINT "_=O=_"
```



Vi sembra che assomigli ad un UFO? Possiamo provare a disegnarne degli altri di forma diversa usando i diversi caratteri e i vari simboli della tastiera. Quest'UFO è stato fatto con la lineetta di sottolineatura (), il segno dell'uguale (=) e la lettera O.

E ora facciamo qualcos'altro:

```
LOCATE 13,10:PRINT "      "
```

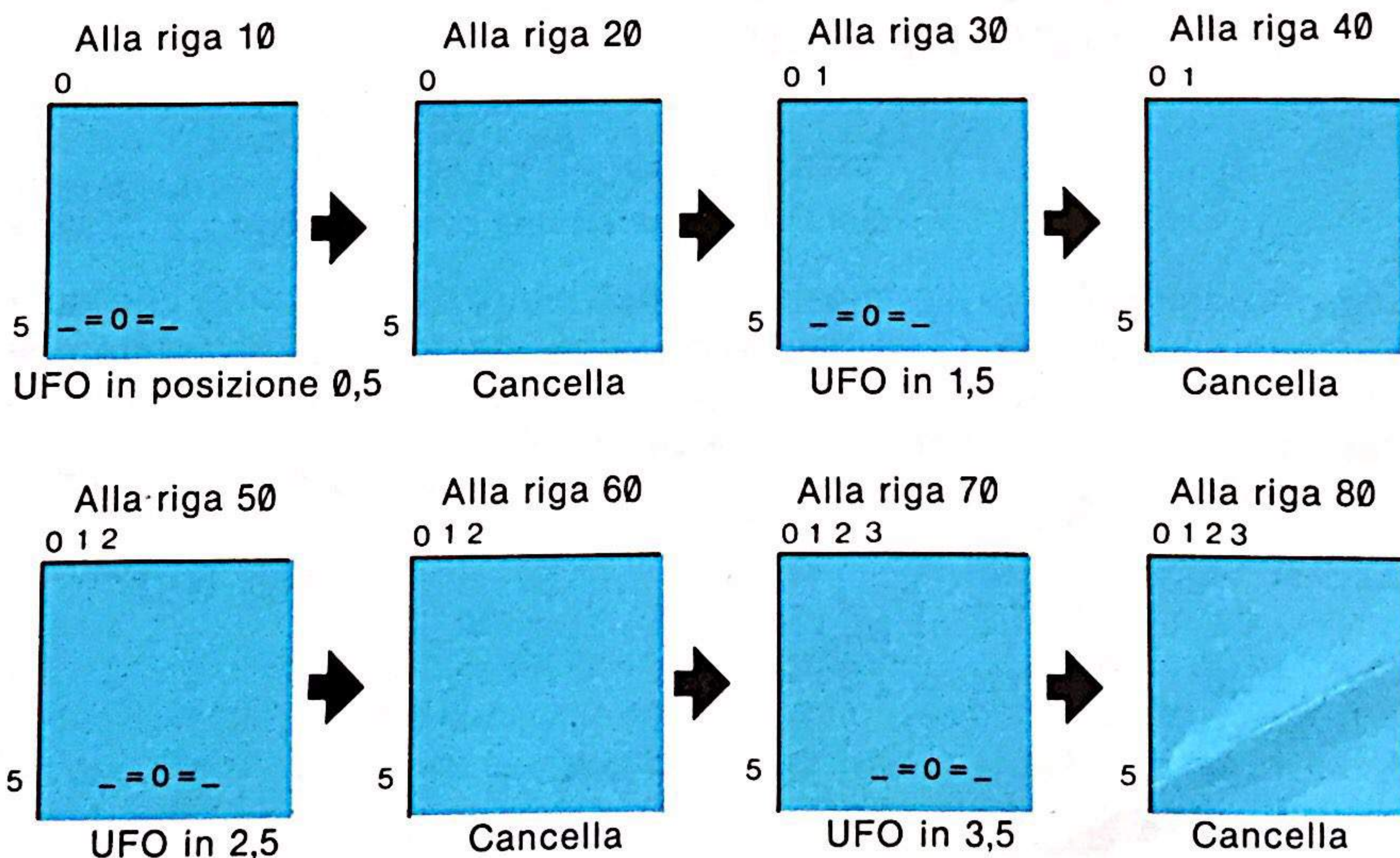
Premiamo `RETURN` e l'UFO scompare! I cinque caratteri sono stati sostituiti da cinque spazi. Lo "spazio" è un modo molto comodo per cancellare delle lettere o delle parole quando non si vuole usare `CLS` per cancellare tutto lo schermo.

VISUALIZZA, CANCELLA, VISUALIZZA, CANCELLA.....

I comandi `LOCATE` e `CLS`, quindi, servono a realizzare un'immagine in movimento. Ecco un programma per far muovere il nostro UFO.

```
5 CLS
10 LOCATE 0,5:PRINT "_=0=_"
20 LOCATE 0,5:PRINT "      "
30 LOCATE 1,5:PRINT "_=0=_"
40 LOCATE 1,5:PRINT "      "
50 LOCATE 2,5:PRINT "_=0=_"
60 LOCATE 2,5:PRINT "      "
70 LOCATE 3,5:PRINT "_=0=_"
80 LOCATE 3,5:PRINT "      "
```

Questi sono i comandi come apparirebbero sullo schermo:



La prima riga, CLS, libera lo schermo, dopo di che ogni coppia di righe—la 10 e la 20, la 30 e la 40, la 50 e la 60 la 70 e l'80—pongono l'UFO in una posizione differente. Se continuiamo in questo modo fino a raggiungere il bordo destro,

	31	32	33	34	35	36	
							0
							1
							2
							3
							4
		-	=	0	=	-	5

gli ultimi comandi saranno

```
650 LOCATE 32,5:PRINT "_=0=_"  
660 LOCATE 32,5:PRINT "      "
```

L'intero programma è composto da 67 righe. Mamma mia! quasi quasi, sarebbe più divertente guardare la televisione!

Non scorraggiamoci. C'è un modo molto più facile. Avrete già notato che i comandi che vengono dopo la riga 5 sono tutti piuttosto simili. Questo è il punto. In effetti, essi sono disposti a coppie quasi identiche l'una all'altra.

```
10 LOCATE 0,5:PRINT "_=0=_"  
20 LOCATE 0,5:PRINT "      "
```

Questa parte non cambia mai
Questa parte è sempre 5
Solo questa posizione cambia: 0,0,1,1,2,2,.....32,32

Ciò che cambia è solo il primo numero dell'indirizzo (posizione), che varia in modo sistematico. Ciò significa che (indovinato?) può diventare una **variabile**. Vediamo così come possono essere utili le variabili: ci permettono di ridurre il programma a sole sette righe.

```
100 CLS  
110 I=0  
120 LOCATE I,5:PRINT "_=0=_"  
130 LOCATE I,5:PRINT "      "  
140 I=I+1  
150 IF I<33 THEN GOTO 120  
160 END
```


Con il comando NEW cancelliamo la memoria del computer e immettiamo questo programma. (Stavolta, abbiamo cominciato a numerare le righe da 100, così che potremo aggiungere altre righe in seguito). Guardate l'UFO: stavolta vola! Esaminiamo il programma riga per riga per capire **perché** vola. La riga 100 ci libera perfettamente lo schermo (Non vogliamo che l'UFO vada a sbattere contro qualcosa). La riga 110 nomina le variabili e assegna loro un valore. Poiché "I" significa posizione sinistra/destra e noi vogliamo iniziare dal lato sinistro, daremo a "I" il **valore iniziale** di zero. Le righe 120 e 130 costituiscono una coppia del tipo 'fa' apparire-cancella' che abbiamo visto poco fa, ma ora usano la variabile "I".

La riga 140 cambia la variabile: dice $I = I + 1$. Attenzione, questa non è aritmetica, è BASIC. Perciò "=" non significa "uguale" ma "prende il valore di". "I" prende il valore di "I + 1" che rende la variabile "I" di un numero più grande prima che andiamo alla riga 150.

La riga 150 significa che se "I" è minore di (<) 33, il computer dovrà tornare alla riga 120. Ora "I" è aumentata di uno e l'UFO apparirà spostato di uno spazio a destra. Quando "I" non sarà più inferiore a 33, il computer non eseguirà "GOTO" 120 ma passerà alla riga 160. A questo punto, il programma si fermerebbe anche senza comando END (in BASIC, END significa 'fine' ed è il comando che termina il programma), perché non ci sono più comandi dopo la riga 150.

Questo programma contiene però una **ripetizione** nella quale il computer continua a ripetere in ciclo le istruzioni dalla riga 120 alla riga 150. Questa ripetizione è controllata dalla condizione che "I" sia minore di 33 e per questo è chiamata **ripetizione condizionale**.

Un modo ancora migliore per farlo

Diamo un'altra occhiata al programma.

```

100 CLS
110 I=0 ← Il valore iniziale è zero
120 LOCATE I,5:PRINT "_=0=_ "
130 LOCATE I,5:PRINT "      "
140 I=I+1 ← Il valore aumenta di uno
150 IF I<33 THEN GOTO 120 ← a ogni ripetizione;
160 END ← la ripetizione si ferma a 33

```

La parte più importante del nostro programma è costituita dalle righe 110, 140 e 150. Esse dicono al computer da quale punto partire (0) e di ripetere il ciclo per 33 volte. Questo tipo di ripetizione è usata spesso in molti programmi e viene eseguita in un modo speciale nel BASIC, con due comandi chiamati **FOR** e **NEXT**


```

100 CLS
110 FOR I=0 TO 32 ←————— Ripeti da 0 a 32
120 LOCATE I,5:PRINT "_=0=_ "
130 LOCATE I,5:PRINT "      "
140 NEXT I ←————— Aumenta di 1 il valore di I;
                    ritorna alla riga 120
150 END

```

Questi due comandi, FOR alla riga 110 e NEXT alla riga 140, sono gli esperti della ripetizione. Hanno bisogno di due sole righe al posto delle tre che erano necessarie prima.

La cosa importante da ricordare è questa formula:

```

FOR variabile = valore iniziale TO valore finale
e
NEXT variabile

```

Per il nostro programma è:

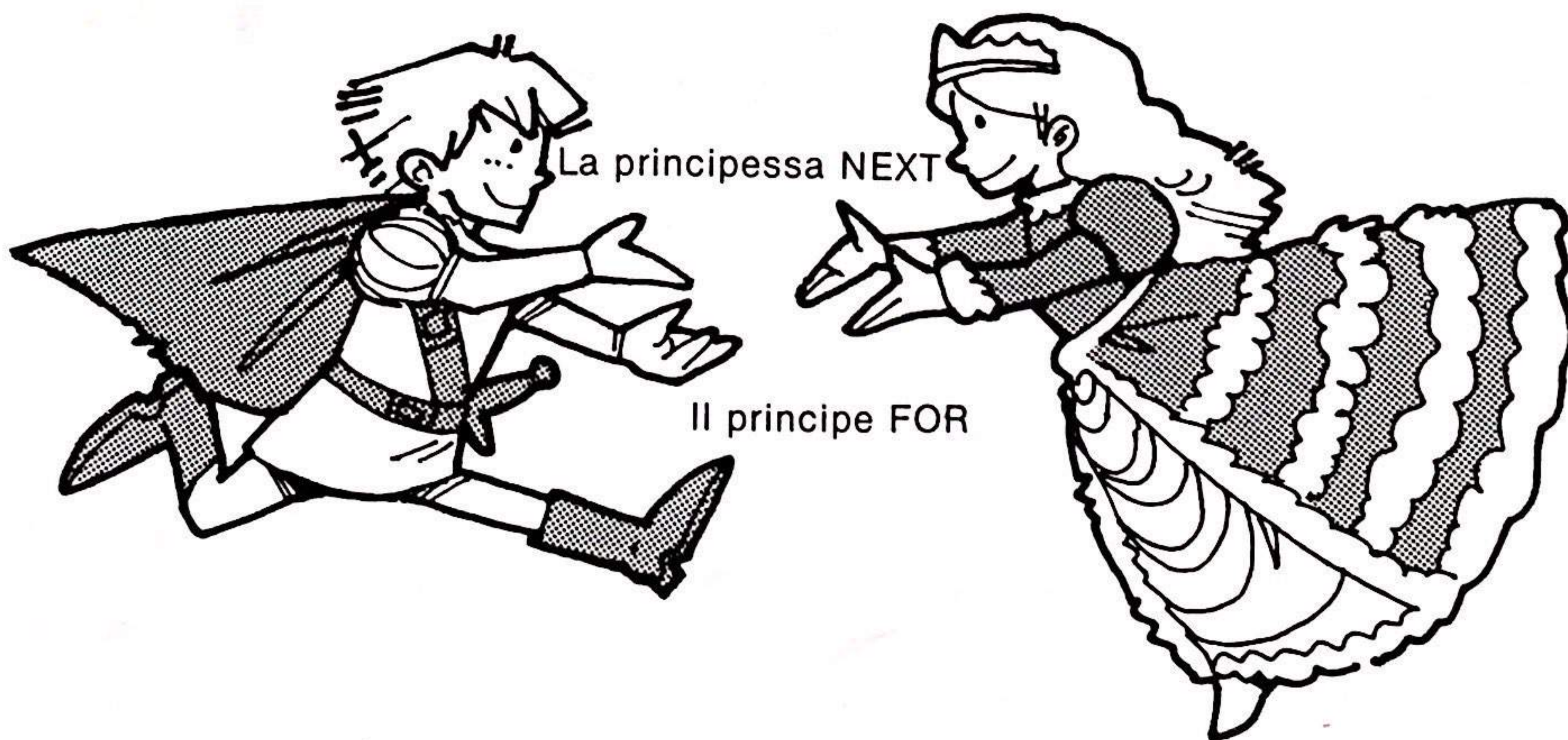
```

110 FOR I=0 TO 32
e
NEXT I

```

FOR dice al computer quante ripetizioni deve fare. NEXT, invece, è un comando molto astuto che conta i passi compiuti e manda il computer alla riga seguente dopo che la variabile ha raggiunto il valore finale.

FOR e NEXT vanno **sempre** in coppia.



Se il principe FOR non riesce a trovare la sua principessa NEXT, il computer diventerà così infelice che farà apparire un messaggio d'errore e si fermerà fino a che non avremo corretto l'errore.

■ METTIAMO TUTTO INSIEME ■

Come...

- Aggiungere colore e suono ai nostri programmi
- Mettere insieme due programmi
- Come realizzare un diagramma di flusso
- Migliorare il nostro programma



AGGIUNGIAMO COLORE E SUONO

Ora siamo pronti ad usare alcune delle funzioni che abbiamo imparato poco fa—per ottenere un UFO colorato, rumoroso, mobile. Immettiamo nel computer questo programma:

```
100 COLOR 15,1:CLS
103 Y=INT(RND(1)*22)
106 C=INT(RND(1)*14)+2:COLOR C
110 FOR I=0 TO 32
120 LOCATE I,Y:PRINT "_=0=_ "
130 LOCATE I,Y:PRINT "      "
133 A=I MOD 12
136 IF A=0 THEN BEEP
140 NEXT I
150 GOTO 100
```

Ripetuto da FOR—NEXT

Abbiamo tutto il tempo che vogliamo per esaminarlo, perché l'intero programma è una ripetizione ciclica. Premiamo però **CTRL** e **STOP** per vedere come funzionano i comandi.

La riga 100 esegue due operazioni: decide il colore (sfondo nero e caratteri bianchi) e libera lo schermo. La riga 103 introduce una nuova variabile, Y, e le assegna un **valore casuale** compreso tra 0 e 21. La riga 106 è molto simile: assegna un valore casuale compreso tra 2 e 15 a una nuova variabile, C, facendola poi diventare un numero di colore (Non vogliamo il colore 1, perché caratteri neri risulterebbero invisibili!).

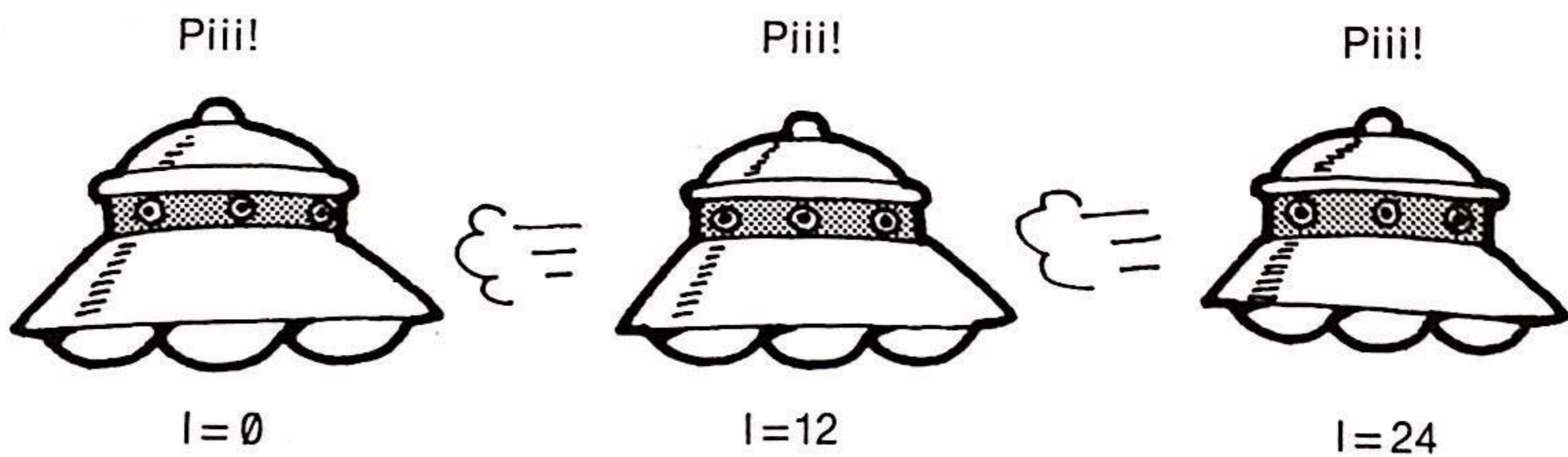
A partire dalla riga 110 troviamo la sequenza FOR—NEXT. Stavolta, la variabile Y è parte dell'“indirizzo” per ogni comando LOCATE. La posizione orizzontale (sinistra/destra) è data da I che varia **regolarmente** da 0 a 32, mentre la posizione verticale (alto/basso) è stata resa **casuale** (RND) nella riga 103: Y può essere un qualsiasi valore compreso tra 0 e 21. Ecco perché quegli UFO di colore diverso appaiono sullo schermo a diverse altezze.

Diamo ora un'occhiata alle righe 133 e 136:

```
A=I MOD 12
IF A=0 THEN BEEP
```

Troviamo due nuove parole. BEEP, naturalmente, significa 'piii!'—il rumore che fa l'UFO. Tuttavia, fa 'piii!' solo se A è uguale a zero e A ha il valore di $I \text{ MOD } 10$. Che cosa dunque significa MOD? MOD è un'abbreviazione di **modulo** e indica il resto che avanza quando un numero viene diviso per un altro. $I \text{ MOD } 10$ è il numero che avanza quando I viene divisa per 10. Se I vale 15, $I \text{ MOD } 10$ è 5, ma se I è 20, $I \text{ MOD } 10$ è 0.

Mano a mano che I aumenta da 0 a 32, $I \text{ MOD } 10$ diventa uguale a 0 tre volte per ogni UFO: alla riga 0, alla 10 e alla 20. Ciò significa che ogni UFO emette tre 'piii!' mentre viaggia attraverso lo schermo.



GIOCO DELL'INDOVINA IL NUMERO + SUONO + VIDEO

Una delle cose più interessanti nella programmazione è mettere insieme due programmi per farne uno. Come vedremo tra poco, mettere insieme due programmi può essere molto utile e divertente.

Eccoli di nuovo:

Programma dell'indovina un numero

```
10 A=INT(RND(1)*5)+1
20 INPUT "Prova a indovinare";B
30 IF A=B THEN PRINT "Azzecato!" ELSE
PRINT "Sbagliato"
40 GOTO 10
```



Programma dell'UFO

```
100 COLOR 15,1:CLS
103 Y=INT(RND(1)*22)
106 C=INT(RND(1)*14)+2:COLOR C
110 FOR I=0 TO 32
120 LOCATE I,Y:PRINT "_=0=_ "
130 LOCATE I,Y:PRINT "      "
133 A=I MOD 12
136 IF A=0 THEN BEEP
140 NEXT I
150 GOTO 100
```

Se ci limitiamo ad immettere i due programmi nel computer così come sono, ci accorgeremo subito che non vanno d'accordo. Il programma girerà fino alla riga 40 e poi tornerà alla 10, di modo che l'UFO non arriverà mai. La prima cosa che dobbiamo fare, quindi, è collegare gli UFO del secondo programma al primo programma. Il posto migliore per collegarli è il punto in cui un numero è stato indovinato correttamente, cioè alla riga 30 (A=B). Scriviamo una nuova riga 30:

```
30 IF A=B THEN GOTO 100 ELSE PRINT "Sbagliato"
```


C'è però un altro problema: come fermare l'UFO dopo che qualcuno ha indovinato il numero giusto. Non vogliamo stare a guardare l'UFO all'infinito, e non vogliamo nemmeno fermare il computer—quello che vogliamo è tornare al nostro gioco. Per far questo, cambiamo la riga 150 nel modo seguente:

```
150 GOTO 10
```

Che ve ne pare di questo programma?

```
10 A=INT(RND(1)*5)+1
20 INPUT "Prova a indovinare";B
30 IF A=B THEN GOTO 100 ELSE PRINT "S
bagliato"
40 GOTO 10
100 COLOR 15,1:CLS
103 Y=INT(RND(1)*22)
106 C=INT(RND(1)*14)+2:COLOR C
110 FOR I=0 TO 32
120 LOCATE I,Y:PRINT "_=0=_ "
130 LOCATE I,Y:PRINT " "
133 A=I MOD 12
136 IF A=0 THEN BEEP
140 NEXT I
150 GOTO 10
```

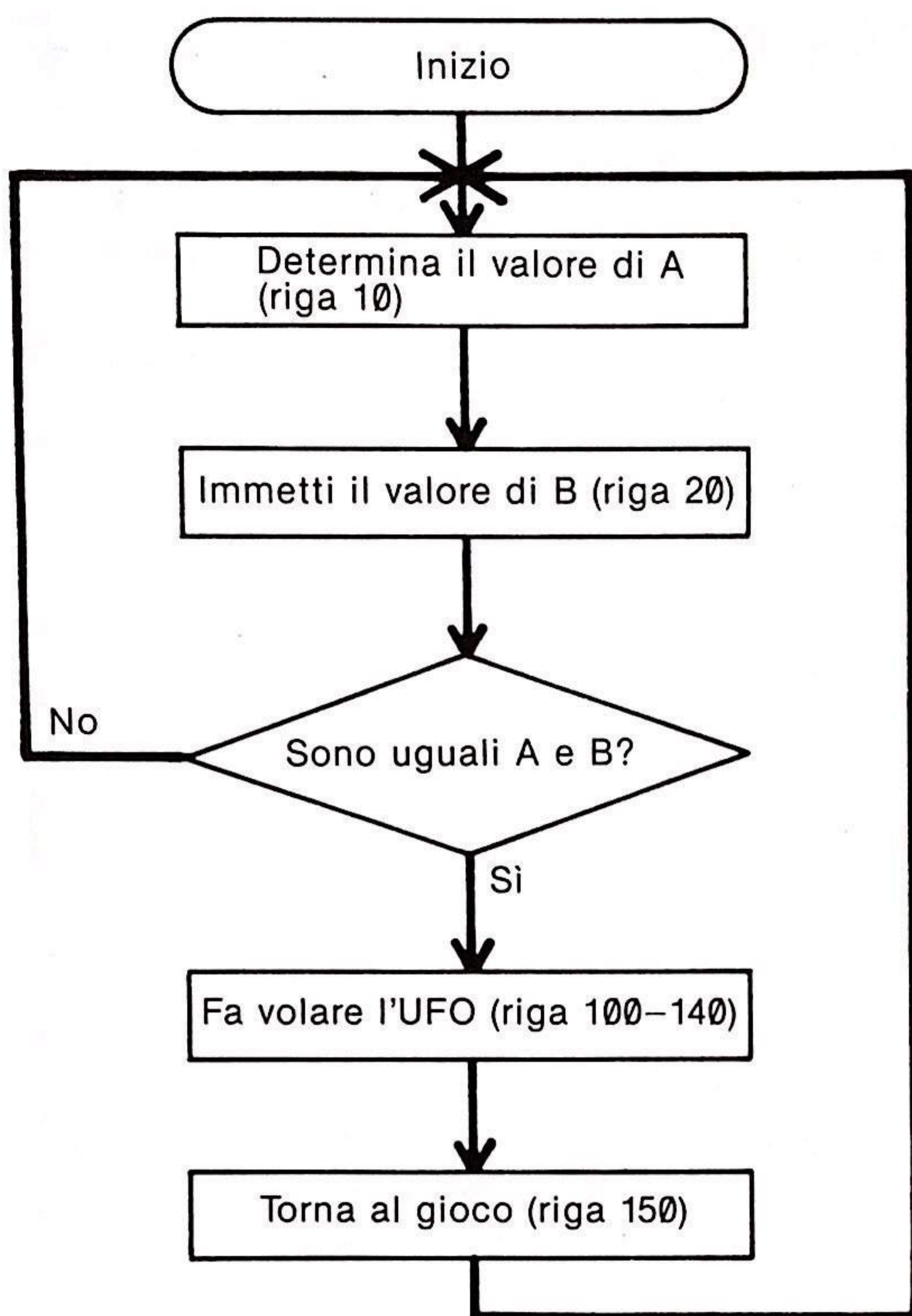
Se "Azzeccato!"
va da UFO

Ritorna a
"Prova a indovinare"

Il diagramma di flusso: un modo di pensare ai programmi

Con le ripetizioni e le funzioni multiple, i nostri programmi sono diventati più lunghi e più complicati. I professionisti di computer, quando compongono programmi complessi, li pianificano accuratamente prima di cominciare a scrivere i comandi. Il modo migliore di progettare un programma è usare un **diagramma di flusso**.

Questo libro non usa i diagrammi di flusso e, quindi, non è necessario capire come funzionano per usare il nostro computer Sony. Se non vi interessano, perciò, potete passare direttamente alla sezione successiva. Tuttavia, tanto per capire che cosa sia un diagramma di flusso, eccone uno per il programma combinato dell'indovina il numero e dell'UFO.



La scatola superiore rappresenta l'inizio del programma e ciascuna delle altre scatole indica un'operazione che ha bisogno di un comando (o di comandi). La linea mostra il corso dell'azione o la logica del programma. Come vediamo, dalla scatola a forma di diamante partono due linee; questa scatola indica che il computer controlla una condizione (IF) per decidere in quale direzione andare. Notiamo inoltre che anche la linea "No" e la linea inferiore formano dei cicli che ripetono il gioco.

Per migliorare il gioco dell'indovina il numero

La differenza tra un programma accettabile (un programma che funziona cioè) e un programma **eccellente** consiste nei miglioramenti che vengono fatti dopo che il programma sia stato scritto. Ecco tre cose che possiamo fare per migliorare il programma dell'indovina il numero.

- Il numero che appare dopo ogni UFO è dello stesso colore dell'UFO. Se tutti i numeri fossero bianchi, non sarebbero più facili da leggere?
- Il gioco dell'indovina il numero si muove su e giù per lo schermo—e comincia sempre sulla riga seguente l'ultimo UFO. Non sarebbe possibile farlo stare fermo?

● Quando il primo UFO comincia a volare, il colore dello sfondo cambia da blu scuro a nero e rimane poi nero. Non sarebbe meglio avere sempre uno sfondo dello stesso colore?

Con poche, semplici aggiunte e alcuni cambiamenti, possiamo risolvere questi problemi.

```
5 COLOR 15,1
7 CLS:LOCATE 0,0 ← Aggiungi
10 A=INT(RND(1)*5)+1
20 INPUT "Prova a indovinare";B
30 IF A=B THEN GOTO 100 ELSE PRINT "S
bagliato"
40 GOTO 10
100 CLS ← Cambia
103 Y=INT(RND(1)*22)
106 C=INT(RND(1)*14)+2:COLOR C
110 FOR I=0 TO 32
120 LOCATE I,Y:PRINT "_=0=_ "
130 LOCATE I,Y:PRINT " "
133 A=I MOD 12
136 IF A=0 THEN BEEP
140 NEXT I
150 GOTO 5 ← Cambia
```

Prima di andare avanti, vediamo se riuscite a capire da soli queste modifiche...

È chiaro? Le nuove righe, la 5 e la 7, risolvono ottimamente tutti i problemi. La riga 5 comanda uno sfondo nero e lettere bianche fin dall'inizio. La riga 7 mantiene sempre il gioco nella parte superiore dello schermo.

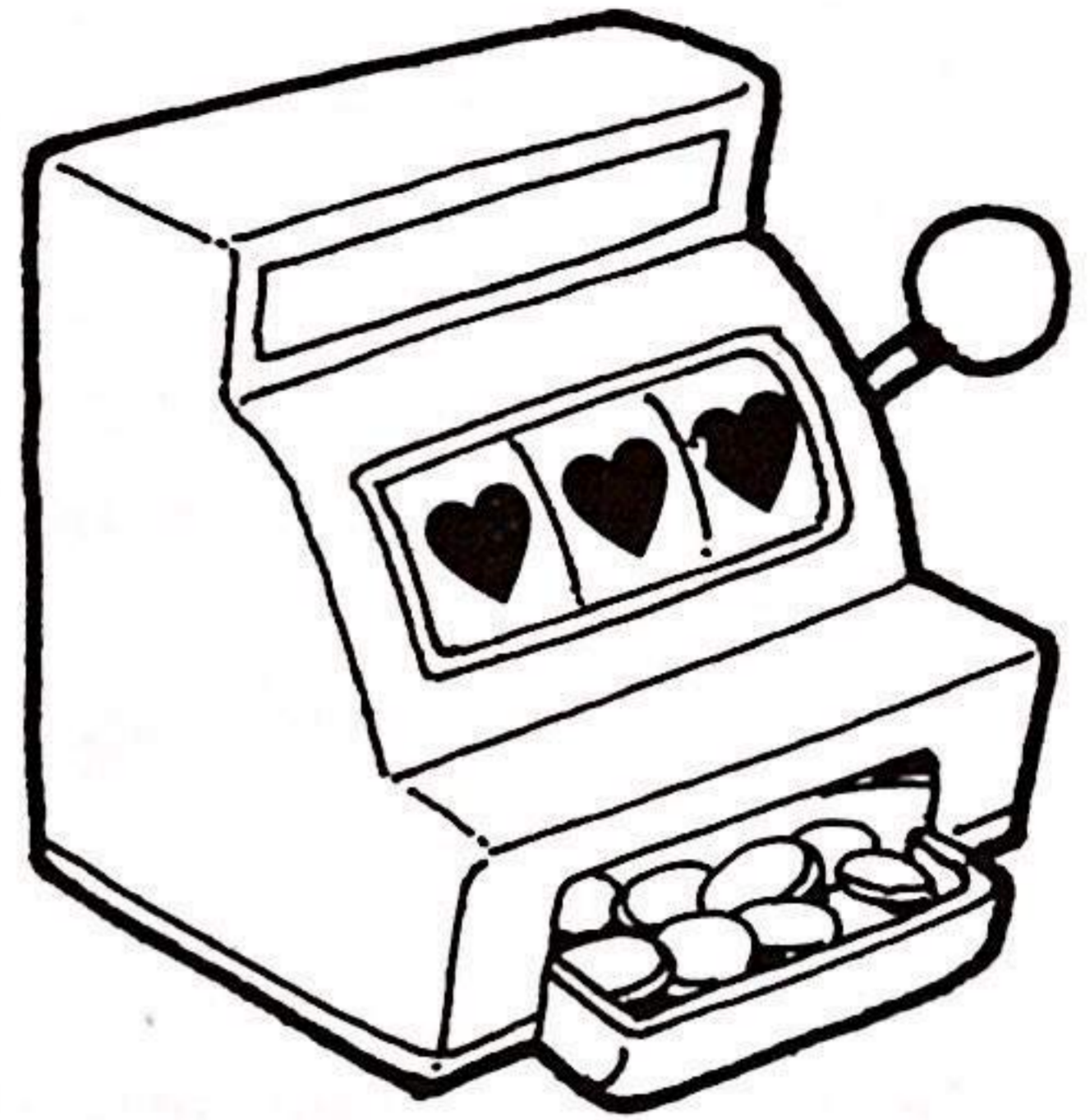
Tuttavia, la cosa più importante da ricordare quando si cambia un programma è questa: **un cambiamento in un posto significa spesso cambiamenti in altri posti**. Nel nostro caso, il comando COLOR alla riga 5 è venuto dalla riga 100, la quale deve essere perciò cambiata e ridotta all'unico comando CLS. Inoltre, poiché il programma inizia ora dalla riga 5 invece che dalla riga 10, bisognerà cambiare anche la riga 150 se vogliamo che le due nuove righe possano girare.

Questo è finalmente un buon programma, un programma che può girare senza problemi. Ne siete completamente soddisfatti? O vi piacerebbe renderlo più interessante? ... Perché gli UFO volano sempre in linea retta? ... Il vostro computer Sony può fare molte, molte altre cose—qualsiasi cosa gli dicitate di fare.

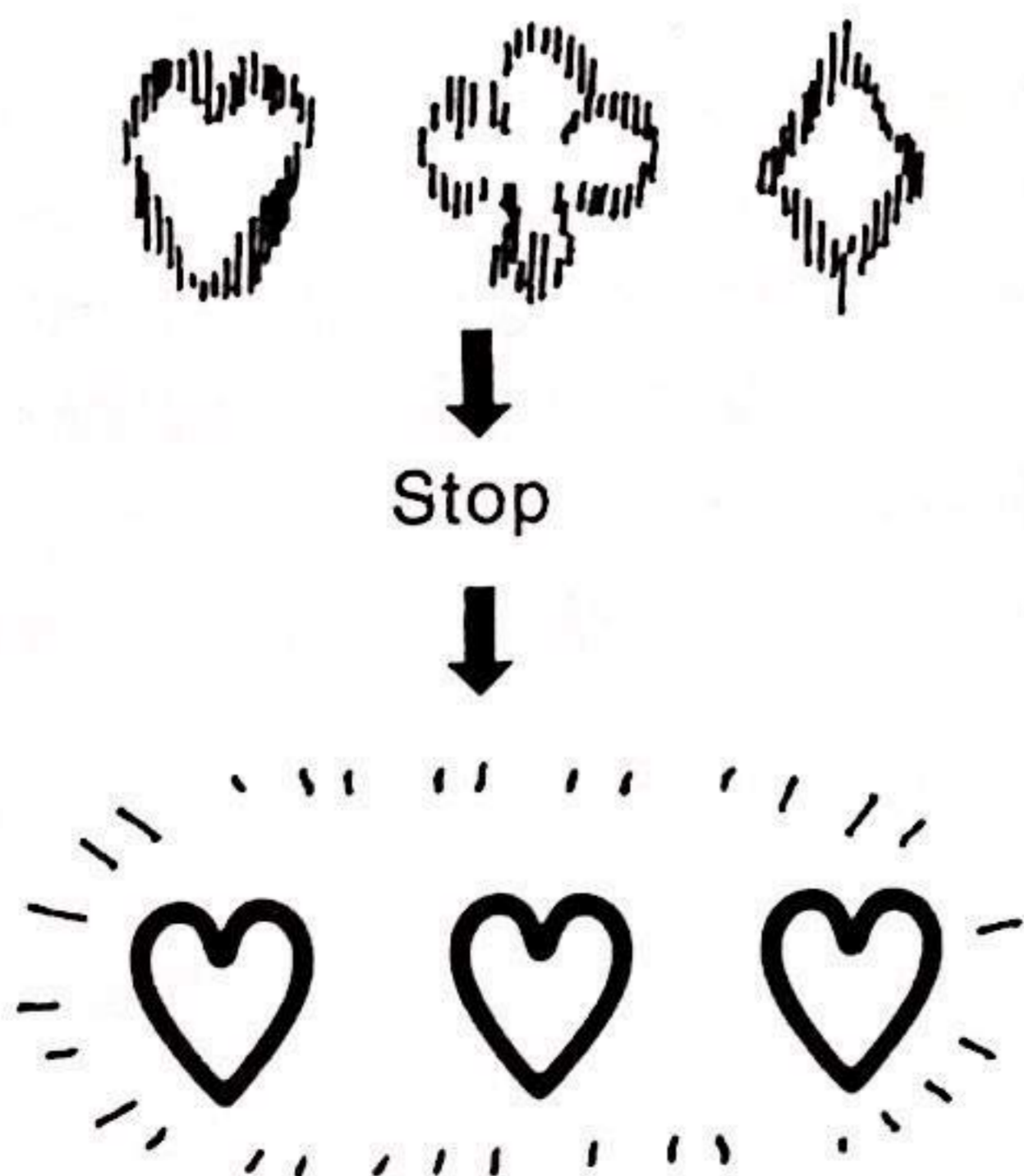
IL GIOCO DELLA MACCHINETTA MANGIASOLDI

Come...

- Programmare una macchinetta mangiasoldi che...
 - Cambi l'immagine sullo schermo fino a che non la fermiamo
 - Confronti le immagini delle finestrelle
 - Conti i punti in base alle nostre scommesse
- Usare una variabile insieme
- Usare il comando ON-GOTO
- Usare una variabile di stringa
- Fermare il programma premendo un tasto
- Stabilire il formato di stampa
- Usare più di una condizione in un comando IF—THEN
- LISTare in sezioni un lungo programma



Nel nostro ultimo programma, ogni volta che indovinavamo un numero venivamo ricompensati dalla comparsa di un UFO volante. Ora, il nostro computer Sony diventerà una “macchinetta mangiasoldi”, stile Las Vegas! Ogni volta che si gioca, si vince o si perde! Non essendo una vera macchinetta mangiasoldi di Las Vegas, il computer non può maneggiare soldi, ma è bravissimo a ricordare e tenere i punti.



Queste sono le regole: prima di tutto, il computer ci chiederà quanti punti vogliamo scommettere. Poi, farà girare rapidamente i simboli contenuti nelle sue tre finestrelle fino a che noi non lo fermiamo. Poi, ci dirà il nuovo punteggio: se i simboli usciti sono tutti e tre uguali, vinciamo il triplo di quello che abbiamo scommesso. Se solo due sono uguali, vinciamo quello che abbiamo scommesso. Se sono tutti e tre diversi, perdiamo il doppio di quello che abbiamo scommesso. Si parte con una **posta** di 100 punti e si vince quando si arriva a 300. Però, se i punti scendono a zero, si perde e il gioco è finito.

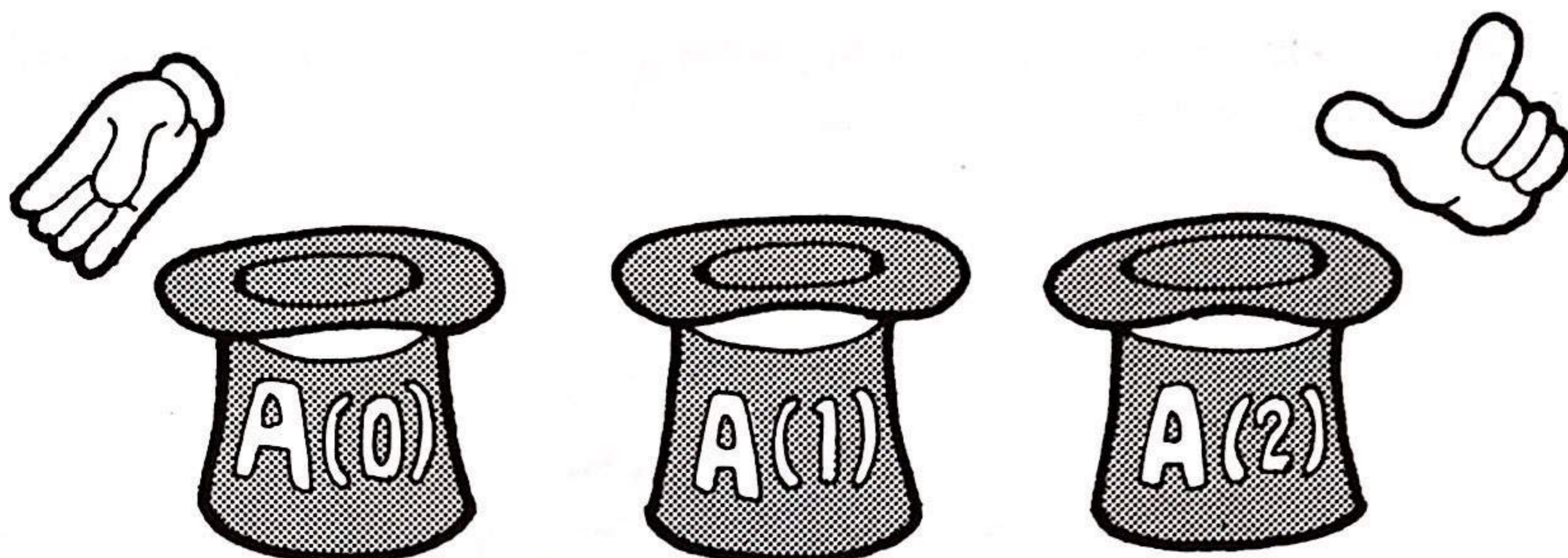
Vi sembra OK? Allora, proviamo a programmarlo!

La parte principale di questo gioco sono le tre finestrelle. In ciascuna di esse appaiono e cambiano continuamente quattro simboli (♥, ♠, ♦, ♣),—così velocemente che non si può vedere quale simbolo ci sia nelle finestrelle, si può solo cercare di indovinarlo. Naturalmente, il nostro programma userà delle variabili per indicare i simboli contenuti in ciascuna finestrella.

COS'È UNA VARIABILE INSIEME? _____

Non è la prima volta che programmiamo delle variabili e perciò sappiamo che una lettera o un nome possono avere un valore variabile. Abbiamo usato nomi come A, B e Q e abbiamo assegnato loro valori che cambiavano con il numero di volte che un ciclo veniva ripetuto, con la posizione di un UFO, o semplicemente, a caso.

Stavolta, abbiamo bisogno di una variabile che possa diventare uno qualsiasi dei quattro simboli che abbiamo visto poco fa. Ognuna delle finestrelle della macchinetta mangiasoldi contiene lo stesso gruppo di simboli, il che significa che possiamo usare la stessa variabile per ognuno dei quattro gruppi. Tuttavia, ogni finestrella funziona indipendentemente dalle altre: a volte si combinano, molto spesso no. In questa particolare situazione, useremo la nostra variabile in tre posti diversi—tre volte la stessa variabile. Ciascuna si chiamerà A, ma sarà accompagnata nello stesso tempo da un numero (tra parentesi), di modo che noi possiamo sapere quale è la variabile di ciascuna finestrella.



$A(0)$, $A(1)$ e $A(2)$ sono **variabili insieme**. Vi sembra complicato? O pensate che sarebbe meglio usare tre nomi diversi? Vi accorgete molto presto che le variabili insieme rendono la programmazione molto più semplice, perché sono molto flessibili.

Il nostro insieme di variabili è **largo** tre variabili ed è **lungo** quanto il numero di valori diversi che gli diamo. Possiamo considerare queste variabili come delle variabili **multi-dimensionali**, parola questa, che ci aiuterà a ricordare il comando BASIC per creare una variabile insieme: **DIM**. In questo programma useremo la riga di comando

```
10 DIM A(2)
```

per creare tre variabili insieme, $A(0)$, $A(1)$ e $A(2)$. (Altri esempi di comandi di variabili insieme sono: **DIM A (10)**, che introduce 11 variabili da $A(0)$ ad $A(10)$; oppure **DIM A(2),B(2)**, che introduce sei variabili—da $A(0)$ ad $A(2)$ e da $B(0)$ a $B(2)$).

C'è un'altra cosa da sapere sulle variabili insieme: anche il numero contenuto tra parentesi può diventare una variabile, come $A(I)$, cosa che le rende ancora più semplici da programmare.

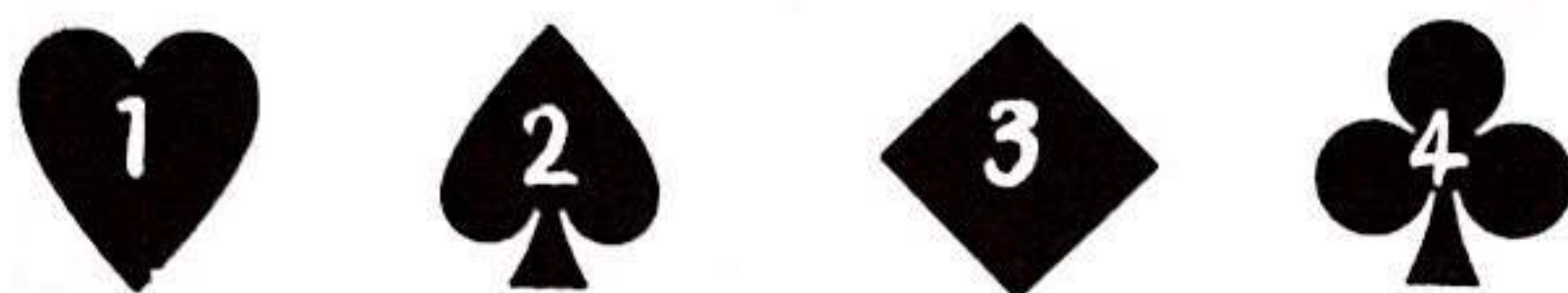
FABBRICHIAMO UNA MACCHINETTA MANGIASOLDI

Vogliamo che le nostre variabili insieme rappresentino simboli diversi—cuori (♥), picche (♠), quadri (♦) e fiori (♣). La prima cosa di cui abbiamo bisogno sono dei numeri di codice per indicare i simboli. Come ricorderete, abbiamo usato dei numeri di codice per rappresentare dei colori e il nostro computer Sony conosce già i numeri di codice dei diversi colori. Stavolta, dobbiamo stabilire un codice per i simboli che useremo.

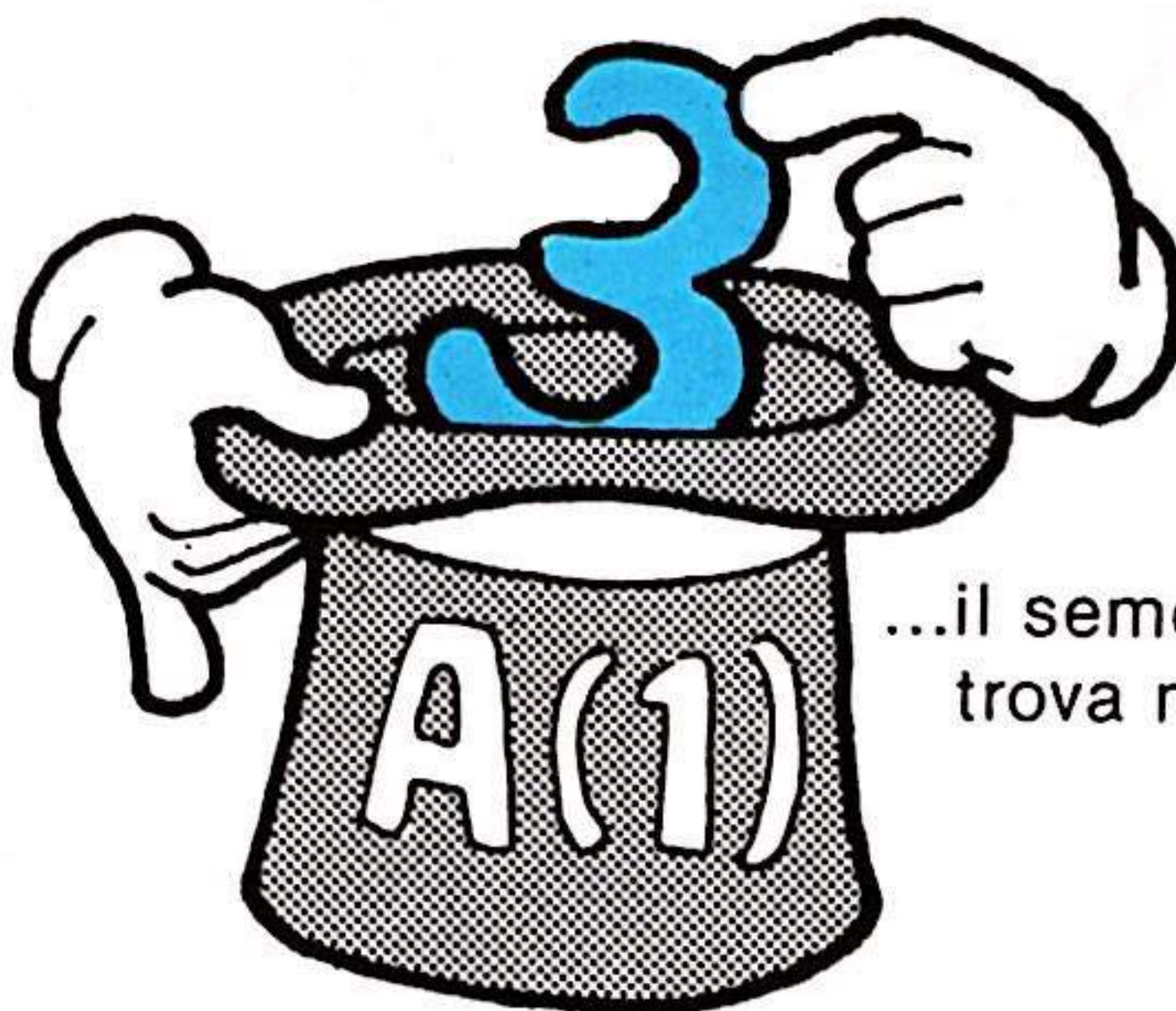
♥—1 ♠—2
♦—3 ♣—4

Ora sappiamo che quando la variabile A(1) assume il valore di 3, nella finestrella (1) ci sarà il seme di quadri.

Prima decidi il codice

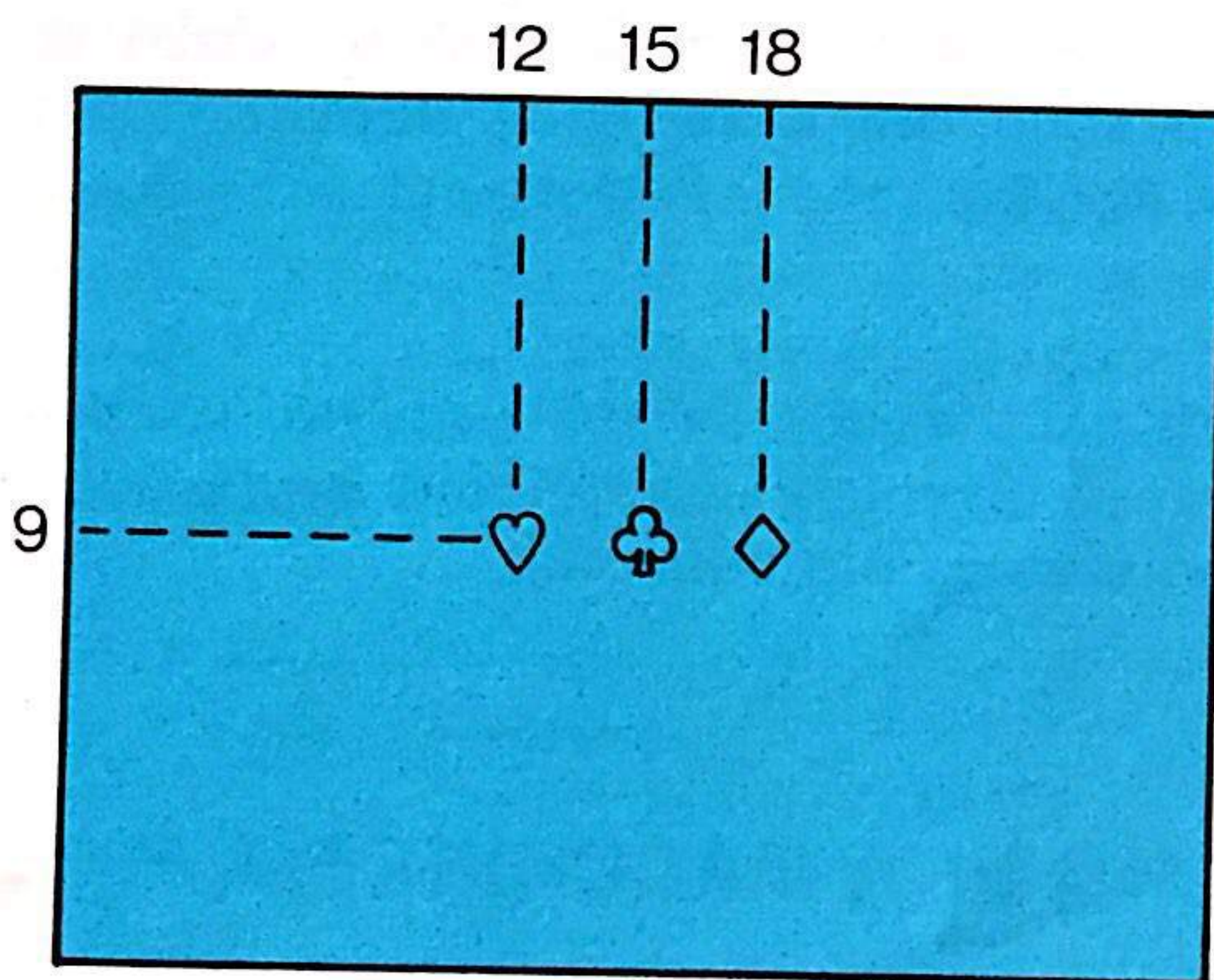


Se succede questo...



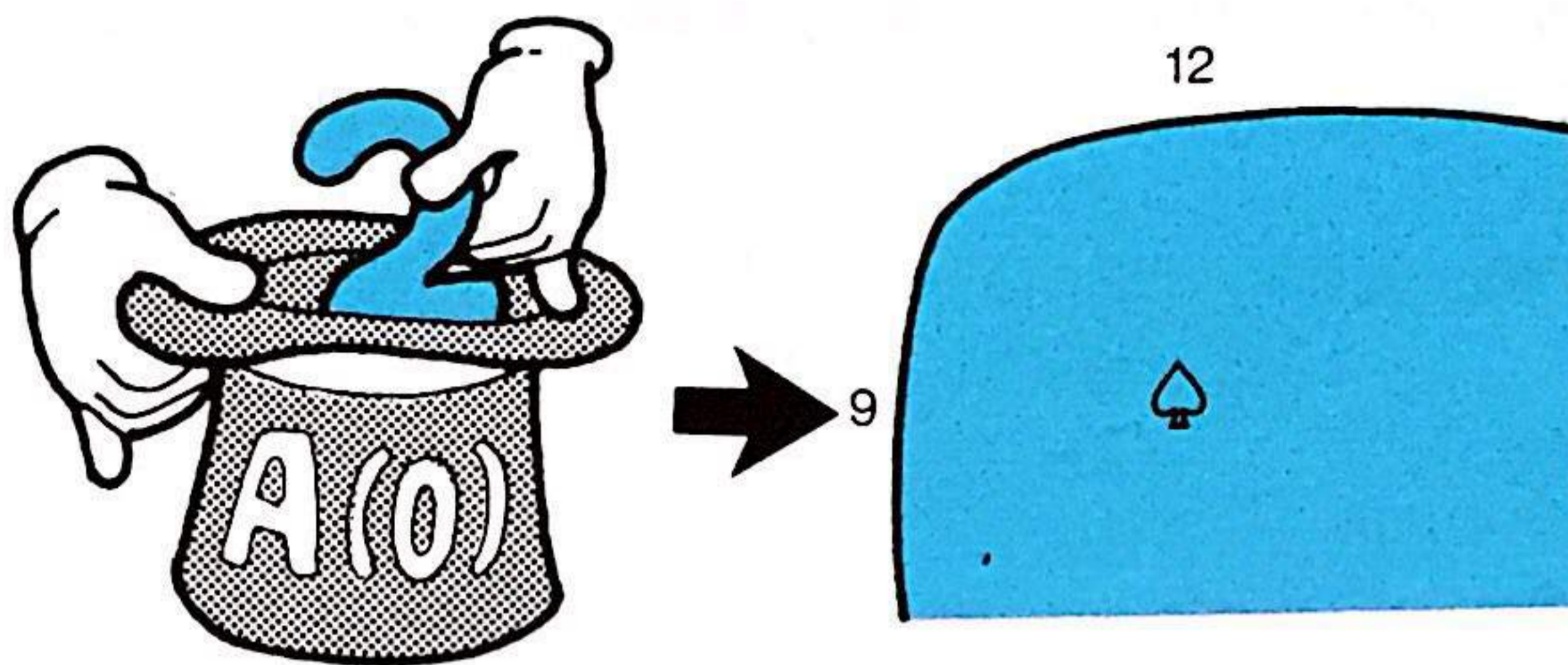
...il seme di quadri (♦) si trova nella finestrella A (1)

Ora dobbiamo decidere la posizione delle finestrelle sullo schermo. Mettiamole in mezzo.

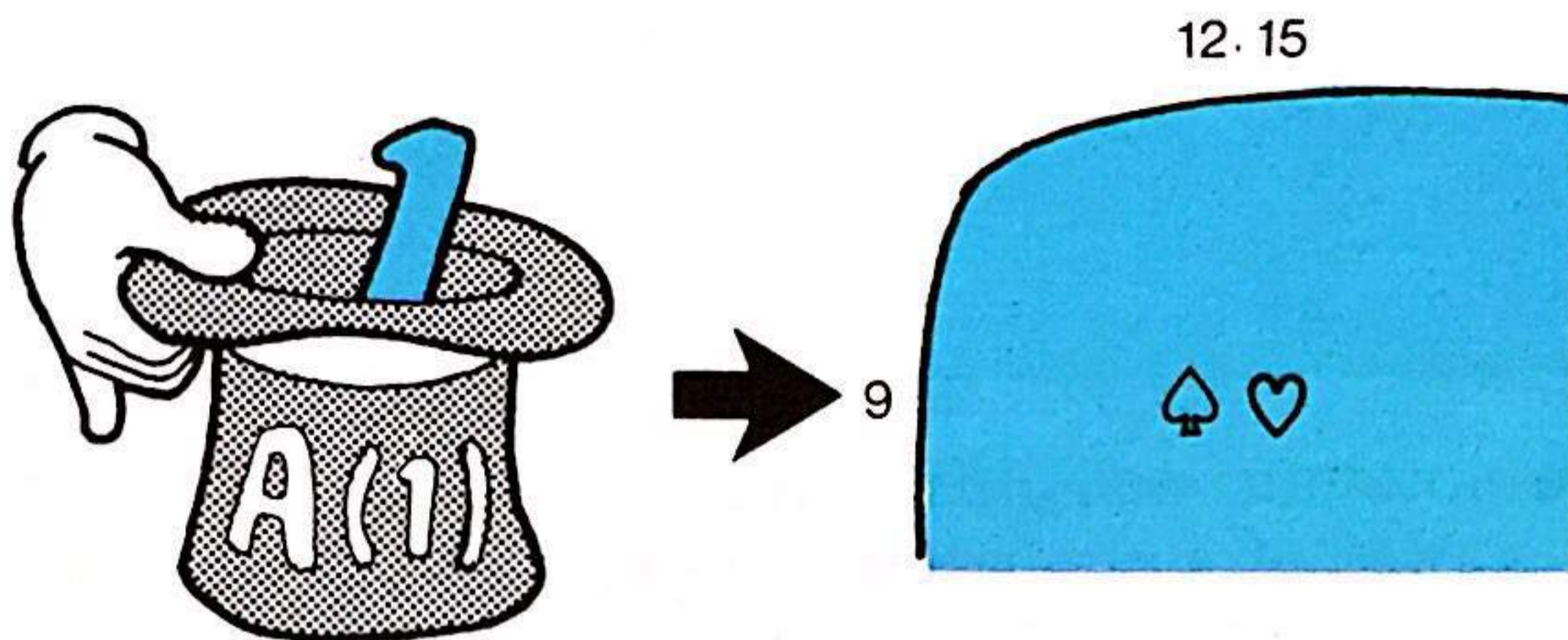


Abbiamo sistemato la finestrella A(0) in 12,9. La finestrella A(1) si trova in 15,9 e la finestrella A(2) in 18,9.

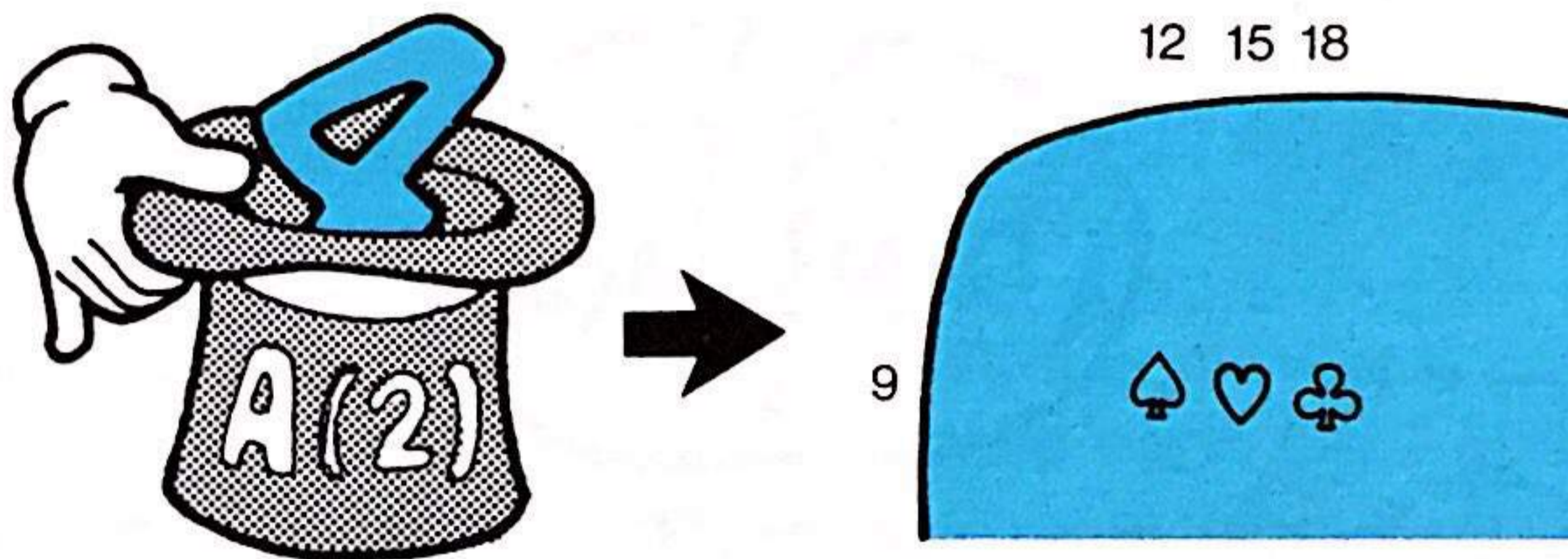
Il progetto del nostro programma per fabbricare una "macchinetta mangiasoldi" è terminato. Proviamo a riesaminare il sistema che abbiamo ideato.



Se $A(0)$ prende il valore di 2, 2 è il codice per ♠, così ♠ appare a 12,9



Se $A(1)$ prende il valore di 1, 1 è il codice per ♥, così ♥ appare a 15,9.



Se $A(2)$ prende il valore di 4, 4 è il codice per ♣, così ♣ appare a 18,9.

Questo sistema farà apparire i simboli nelle finestrelle. Siccome ogni variabile cambia molto velocemente—velocemente quanto cambiavano le posizioni degli UFO nel nostro precedente programma—i simboli contenuti nelle finestrelle cambieranno così in fretta che non avremo il tempo di vederli chiaramente.

Abbiamo fatto un piano del nostro programma e questo piano ci è chiaro. È giunto ora il momento di dare i comandi.

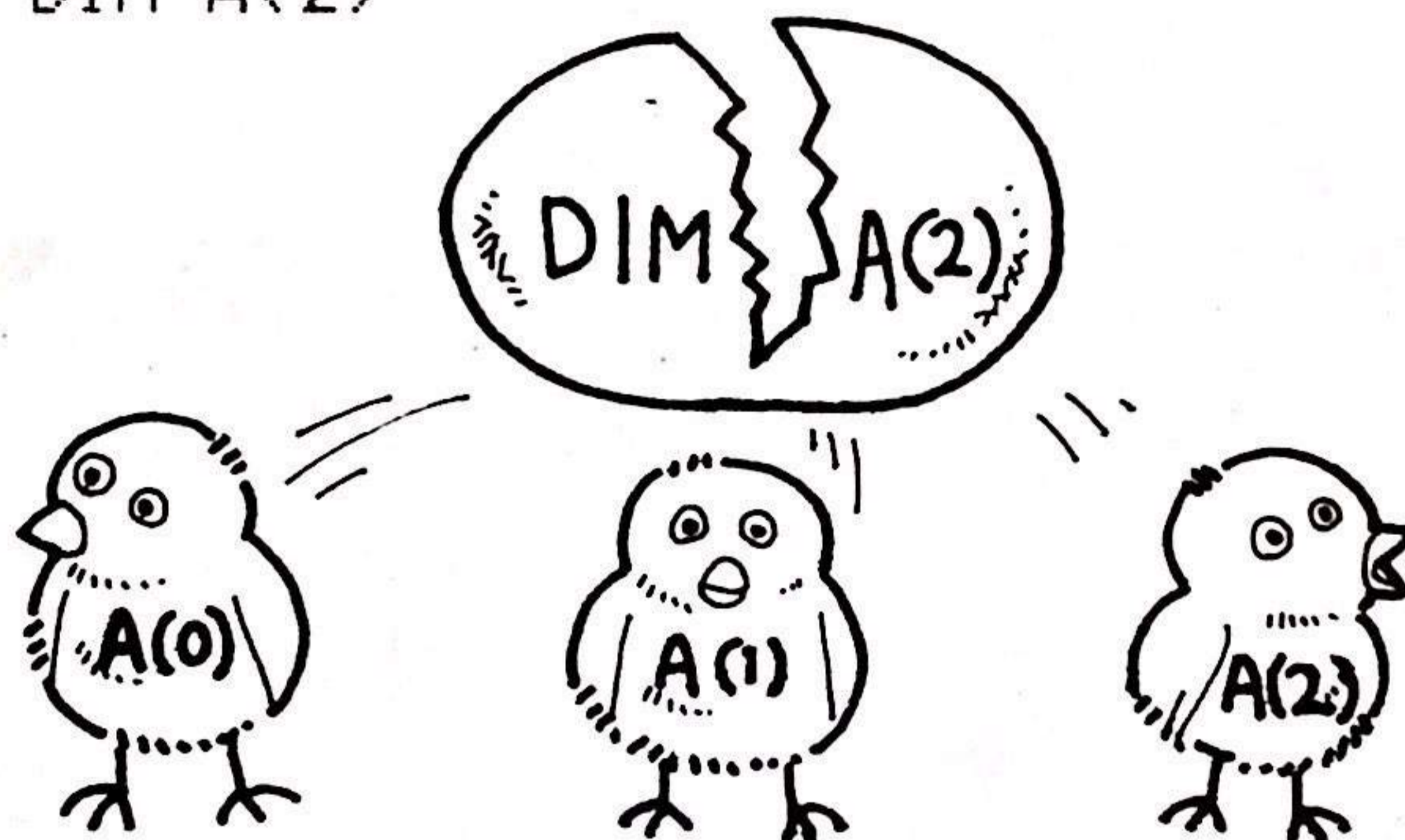

```

10 DIM A(2)
20 CLS
30 FOR I=0 TO 2
40 A(I)=INT(RND(1)*4)+1
50 LOCATE I*3+12,9
60 ON A(I) GOTO 70,80,90,100
70 PRINT "♥":GOTO 110
80 PRINT "♠":GOTO 110
90 PRINT "♦":GOTO 110
100 PRINT "♣"
110 NEXT I
120 GOTO 30

```

Come abbiamo spiegato nell'ultima sezione, la riga 10 introduce tre variabili insieme. Ogni volta che usiamo delle variabili insieme, dobbiamo dirlo al computer all'inizio del programma.

```
10 DIM A(2)
```



La riga 20, naturalmente, libera lo schermo. La riga 30 comincia con FOR, il che significa che se c'è un comando FOR, ci sarà anche un comando NEXT in qualche parte del programma. La sezione FOR—NEXT va dalla riga 30 alla riga alla riga 110 e i comandi contenuti in questa sezione verranno ripetuti in un ciclo continuo. Dopo il comando FOR, la riga 30 crea una nuova variabile I, che useremo tra parentesi nelle nostre variabili insieme.

Vi ricorda niente la riga 40? È lo stesso tipo di comando di **numero casuale** che abbiamo usato nel gioco dell'indovina il numero.

Quando la variabile I prende il **valore iniziale** di 0, la riga 40 diventerà

$$A(0) = \text{INT}(\text{RND}(1) * 4) + 1$$

Ciò significa che $A(I)$ può essere qualsiasi numero compreso tra 0 e 4, giusto? $A(I)$ è la finestrella di sinistra e i numeri 1-2-3-4 sono i numeri di codice dei simboli ♥, ♠, ♦, ♣. In altre parole, quando I prende il valore di 0, la riga 40 dice al computer di scegliere uno dei simboli per la finestrella sinistra.

Ora, dobbiamo dire al computer in quale posizione dello schermo far apparire le tre finestrelle.

```
50 LOCATE I * 3 + 12,9
```

Per la finestrella sinistra, quando I ha il valore di 0, il cursore si porterà nella posizione 12,9. Quando I ha il valore di 1, per la finestrella centrale, la posizione diventerà 15,9. Per la finestrella di destra, I avrà il valore di 2 e la posizione sarà $2 \times 3 + 12,9$ —o 18,9.

Il computer ha scelto un simbolo e il cursore si è portato sulla finestrella. Il computer, tuttavia, non farà apparire il simbolo fino a che non gli diremo di farlo. La riga 60 è molto importante perché è quella che ci permette di dirglielo.

```
60 ON A(I) GOTO 70,80,90,100
```

Qui troviamo il comando GOTO per quattro righe diverse nello stesso tempo—la 70, l'80, la 90 e la 100. Come fa il computer a sapere a quale riga andare? La decisione viene presa dalla parte ON A(I) di questo comando. Questa è la forma che viene sempre usata per un comando ON-GOTO:

```
ON qualcosa GOTO riga 1, riga 2, riga 3,
```

Questo significa per esempio: se qualcosa è uguale a 1, va' alla riga 1; se qualcosa è 2, va' alla riga 2; se qualcosa è 3, va' alla riga 3 ... In altre parole, il valore di "qualcosa" decide automaticamente a quale riga il computer dovrà andare.



Per fare un esempio, diciamo che $A(0)$ ha il valore di 3 (il numero di codice di \spadesuit). La riga 60 dice al computer di andare alla riga 90 quando la variabile assume questo valore.

```
90 PRINT "♠":GOTO 110
```

Cosa succede se 2 è il valore della variabile $A(0)$? In questo caso la riga 60 manda il computer a

```
80 PRINT "♣":GOTO 110
```

Dovrebbe essere chiaro ora in che modo il computer sceglie uno dei quattro simboli e lo fa apparire nella finestrella.

Dopo ogni comando PRINT, il computer va alla riga 110. (Alla riga 100 non c'è nessun comando GOTO 110 perché il computer va automaticamente alla riga successiva).

La riga 110 dice al computer di cambiare il valore di I da 0 a 1. Poi, il computer torna alla riga 40. Sapete che cosa succederà dopo? Pensate un po' al programma e cercate di capirlo da soli prima che lo spieghiamo.

OK. Quando I è uguale a 1, la riga 50 diventa

```
LOCATE 1 * 3 + 12,9
```

che significa LOCATE 15,9. In altre parole, la posizione dove il simbolo verrà fatto apparire si è spostata da 12,9 a 15,9—dalla finestrella di sinistra, cioè, alla finestrella centrale. Uno dei simboli apparirà quindi nella posizione 15,9. Poi, il computer cambierà il valore di I da 1 a 2 e tornerà alla riga 40. Stavolta, la posizione diventerà 18,9—cioè, la finestrella destra. Il computer ha ripetuto il programma tre volte (in meno di un secondo!).



Questo è quello che potrebbe apparire sullo schermo, ma non si può mai sapere quali simboli il computer sceglierà a caso.

Le righe dalla 30 alla 120 formano un ciclo. Il computer girerà e girerà, usando le variabili insieme per scegliere e far apparire i simboli nella finestra sinistra, in quella centrale, in quella destra, e poi ancora sinistra, centro, destra ... Poiché il programma contiene un ciclo che si ripete all'infinito, i simboli sembreranno girare continuamente sullo schermo, proprio come farebbero in una vera macchinetta mangiasoldi di Las Vegas.

Se non avete ancora provato questo programma, provate ora a introdurlo nel computer. Controllate bene che non ci siano errori. Quando darete il comando RUN, avrete la conferma che siamo effettivamente riusciti a creare una macchinetta mangiasoldi usando le variabili insieme. (Nel modo SCREEN 0, comunque, il bordo destro di ogni simbolo non è visibile.)

Ma, come possiamo interrompere questo ciclo senza fine e sapere il punteggio? Per trasformare la nostra macchinetta mangiasoldi in un gioco, dobbiamo scrivere qualche altro comando nel nostro programma. Per far ciò, useremo alcune variabili che, invece di valori numerici, assumono delle lettere e sono chiamate **variabili di stringa**. Per queste, ci vuole una spiegazione speciale.

VARIABILI DI STRINGA

Abbiamo imparato che una variabile può assumere il valore di qualsiasi **numero**—come 1, 2, 3, 191 o 252. Le variabili possono assumere anche il “valore” di qualsiasi **lettera**—o di un gruppo di lettere che costituiscono una **stringa**.

Facciamo qualche esercizio per vedere in che modo una variabile viene usata con il valore di una lettera o di una stringa. Battiamo **PRINT A\$** e premiamo **RETURN**. (\$ si trova sul tasto del numero 4 e va perciò battuta insieme con il tasto **SHIFT**).

```
PRINT A$
```

```
OK
```



Non è successo niente. Il computer non ha emesso nessun "piii!", né alcun messaggio d'errore, né ha stampato niente. Ora immettiamo questo comando:

```
A$="ABC"  
PRINT A$
```

Premiamo `RETURN` e sullo schermo apparirà:

```
A$="ABC"  
OK  
PRINT A$  
ABC  
OK  
■
```

Come possiamo vedere, A\$ viene usata come una variabile e il suo valore è "ABC". Confrontate questo comando con quest'altro:

```
A=345  
OK  
PRINT A  
345  
OK  
■
```

Lo conosciamo già, no? La variabile A assume il valore di 345 e non è seguita dal segno \$ (segno del dollaro).

Avete già capito la regola? **Quando una variabile termina con il segno \$, essa ha il valore di una lettera o di un gruppo di lettere (chiamato "stringa"). Se non c'è il segno \$ alla fine di una variabile, essa prende il valore di un numero.**



C'è un'altra cosa da ricordare: il valore deve essere posto tra **virgolette**, in questo modo:

```
A$="ABC"
```

Proviamo a fare insieme questi esempi, per capire meglio la differenza tra le variabili numeriche e le variabili di stringa.

A=123	A prende il valore di 123
OK	
B=234	B prende il valore di 234
OK	
PRINT A+B	Fa apparire A+B
357	La somma è 357
OK	
A\$="123"	A\$ prende il valore di "123"
OK	
B\$="234"	B\$ prende il valore di "234"
OK	
PRINT A\$+B\$	Fa apparire A\$+B\$
123234	La somma è "123234"
OK	
A\$=987	A\$ prende il valore di 987
Type mismatch	Piiii!! Non è possibile
OK	
■	

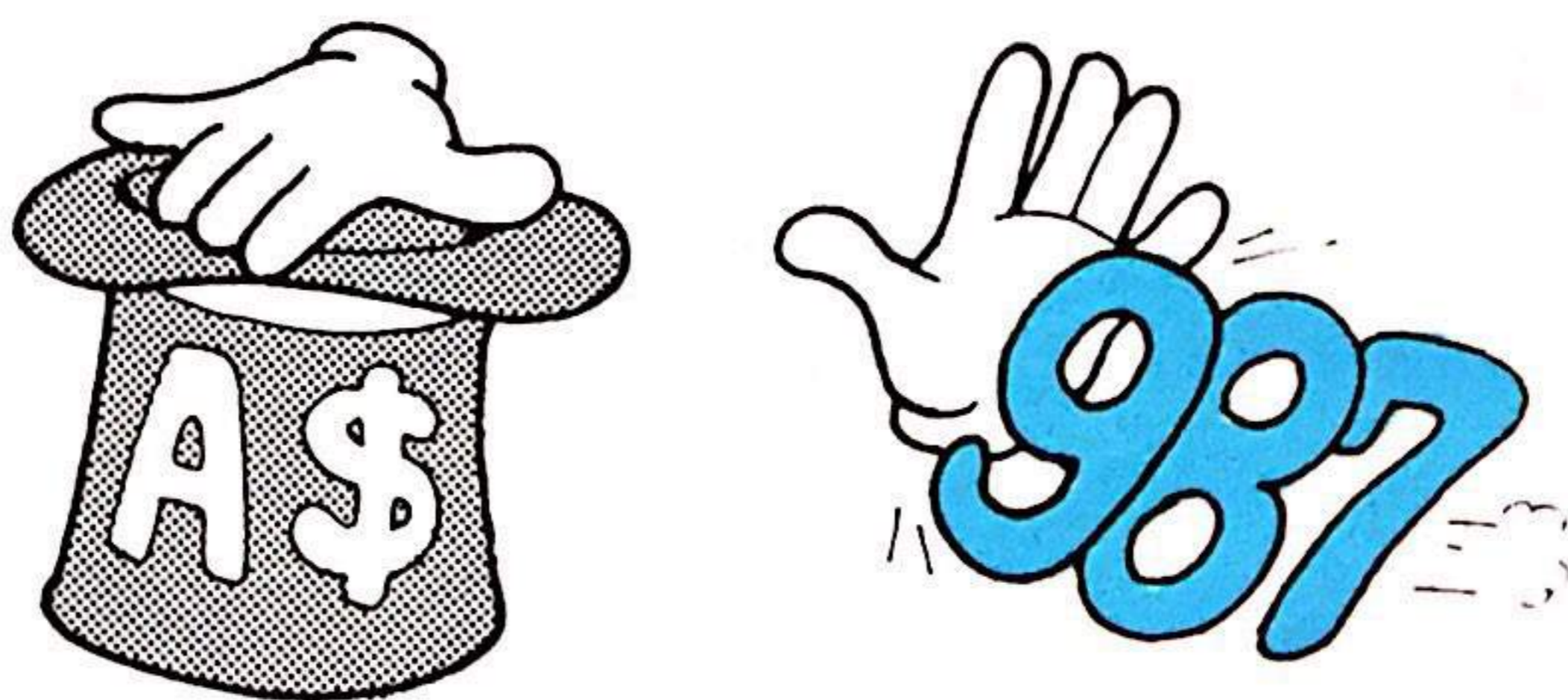
Quando immettiamo `A$="123"`, il valore della variabile `A$` diventa 123—non il numero centoventitre, ma i **caratteri** uno-due-tre. Nella riga successiva, `B$` prende il valore di 234 (caratteri due-tre-quattro). Perché? Perché abbiamo usato le virgolette e il segno \$. Essi hanno detto al computer che noi stiamo usando una variabile di stringa, di modo che 123 viene letto non come un numero, ma come un gruppo di caratteri, come se 1-2-3 fossero delle lettere.

Quando il computer somma `A$+B$`, li unisce, formando un unico gruppo di caratteri. Un esempio analogo potrebbe essere `"ABC"+"BCD"="ABCBCD"`.

Quando abbiamo immesso `A$=987`, il computer ci ha risposto con un messaggio d'errore.

Type mismatch (abbiamo battuto due cose incompatibili tra di loro)

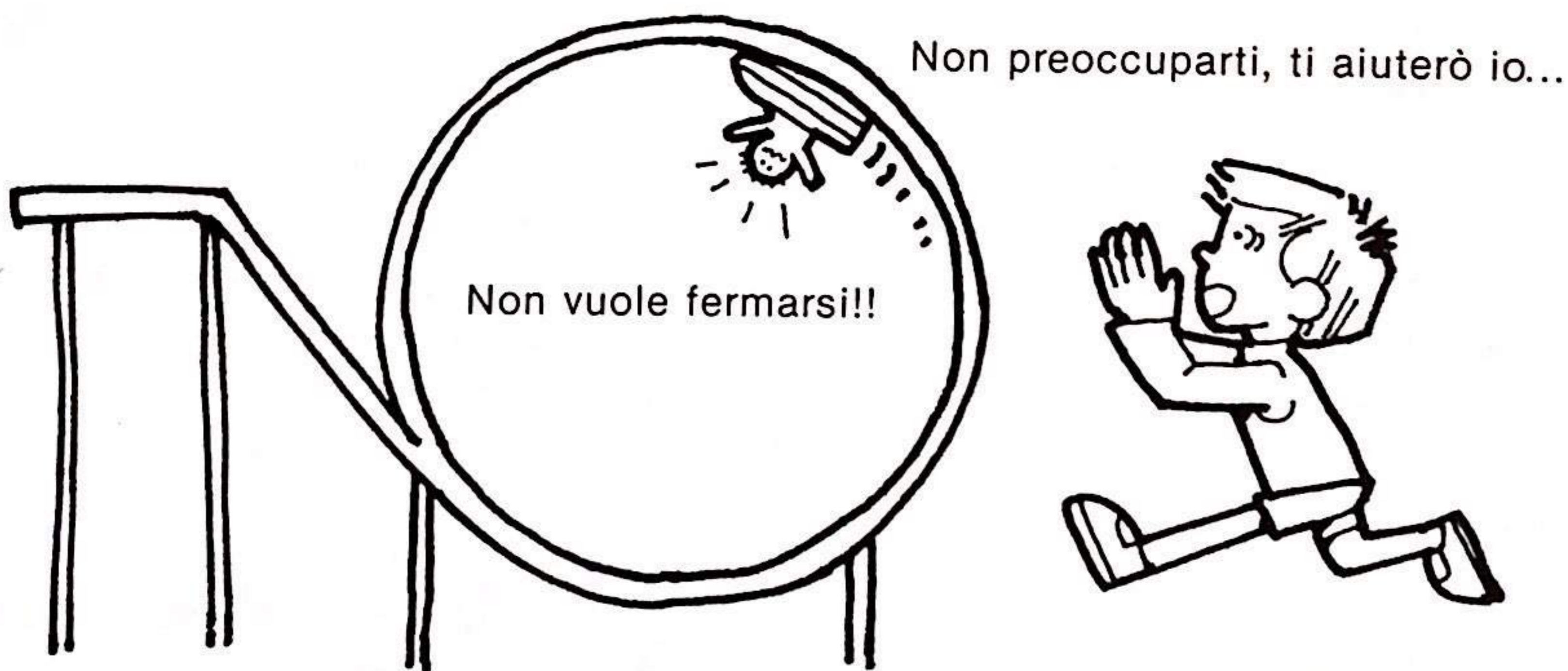
Senza le virgolette, 987 è un numero (novecentottantasette) che è **incompatibile** con la variabile di stringa e il segno \$.



Ora sappiamo che quando il nome di una variabile termina con \$, tale variabile è una variabile di stringa alla quale va assegnato il valore di una lettera o di un gruppo di lettere, valore che, a sua volta, deve essere scritto "tra virgolette". Comunque, torniamo al nostro programma per il gioco della macchinetta mangiasoldi.

COME FERMARE UN CICLO CON INKEY _____

Il ciclo FOR-NEXT che abbiamo usato nel gioco dell'indovina il numero si era fermato da solo quando la variabile aveva raggiunto l'ultimo valore. Il comando che troviamo alla riga 120, invece,—il comando GOTO 30— significa che il ciclo del gioco della macchinetta mangiasoldi non si fermerà mai.



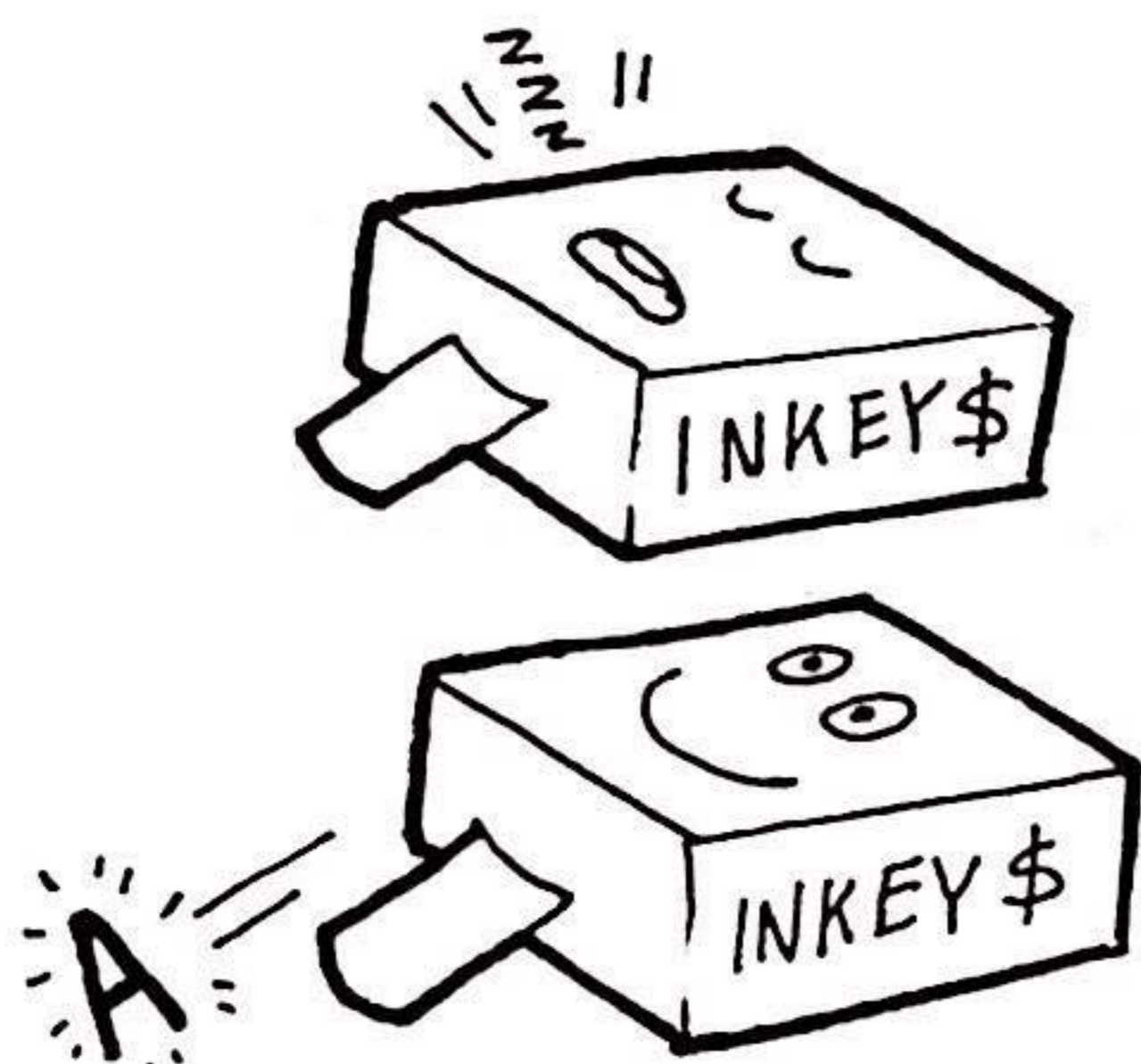
Naturalmente, si può sempre fermare il computer premendo i tasti **CTRL** e **STOP**. Così facendo, però, si fermerebbe l'intero programma senza contare che, una volta riavviato il programma con il tasto RUN, avremmo di nuovo un ciclo che si ripete all'infinito. Invece, noi vogliamo fermare la macchinetta e guardare i simboli, per vedere se sono uguali e sapere così quanti punti abbiamo vinto o perso. E poi vogliamo che il programma riparta, in modo da riprendere il gioco.

Il modo migliore per fermare la macchinetta mangiasoldi è aggiungere queste due righe al nostro programma:

```
103 K$=INKEY$
106 IF K$="A" THEN END
```

K\$ è una variabile di stringa, naturalmente. Alla riga 106 possiamo vedere che quando il valore di K\$ è A, la macchinetta mangiasoldi smetterà (END) di far girare i simboli nelle finestrelle.

Che cosa significa **INKEY\$**? È una funzione del linguaggio BASIC. INKEY significa **introdurre** (in-) dalla tastiera (**key-board**)—un qualsiasi tasto noi premiamo. La riga 103, quindi, significa che **K\$ assume il valore di qualsiasi tasto alfabetico venga premuto**. Se non premiamo nessun tasto, il computer andrà avanti nell'elaborazione del programma senza assegnare nessun valore a K\$. Se premiamo **B** o **C** o **X** il computer continuerà a far girare i simboli della macchinetta, ma se premiamo il tasto **A** (in maiuscolo) la riga 106, comanderà al computer di fermarsi.



Se non premete nessun tasto, io non faccio niente

Se premete il tasto **A**, darò il valore A

Il ciclo FOR—NEXT e GOTO 30, che va dalla riga 30 alla riga 120, contiene un comando END (fine). Il computer, tuttavia, non ascolterà tale comando (non fermerà la macchinetta) fino a che non glielo diciamo premendo il tasto **A**. In altre parole, la macchinetta mangiasoldi continuerà a far girare e a cambiare i suoi simboli fino a che non le diremo di fermarsi.

```
30 FOR I=0 TO 2
```

```
103 K$=INKEY$
106 IF K$="A" THEN END
110 NEXT I
120 GOTO 30
```

Ciclo FOR—NEXT che visualizza 3 simboli nelle finestrelle

Il programma si fermerà (END) se viene premuto il tasto **A**.

QUAL È IL PUNTEGGIO?

Quando la macchinetta mangiasoldi smette di far girare i simboli, vogliamo sapere quanti punti abbiamo vinto (o perso!). Questo significa che dobbiamo scrivere nel programma anche le regole del gioco. Vi ricordate le regole? Si comincia con una **posta** di 100 punti. Poi si **scommettono** alcuni dei punti che si hanno—un numero qualsiasi compreso tra 1 e 100. Aggiungiamo questa informazione al programma.

```
23 P=100:LOCATE 4,18:PRINT USING "POSTA:####";P
26 LOCATE 0,20:INPUT "SCOMMESSA";B
```

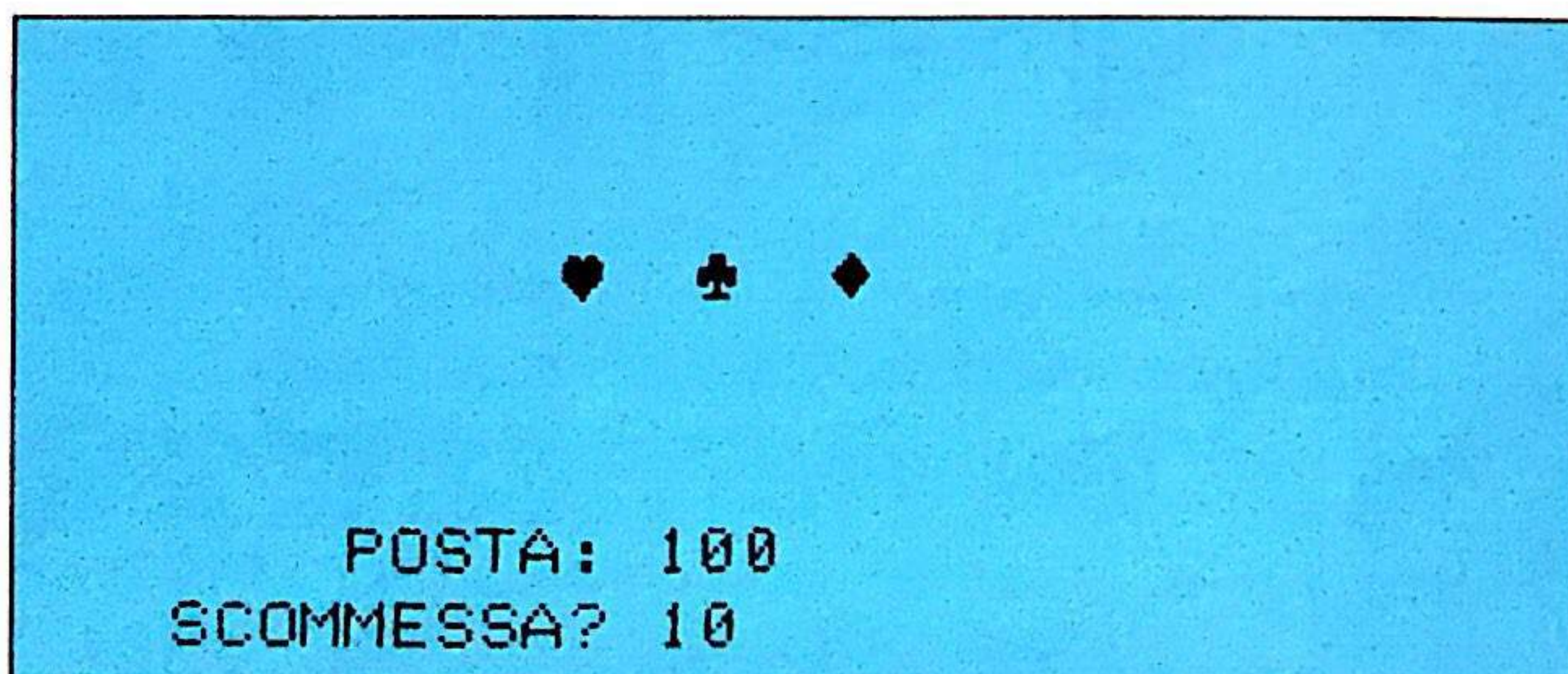
Consideriamo prima la riga 23. La posta iniziale cambierà ogni volta che scommettiamo, perciò useremo una variabile, che chiameremo P. La posta P ha un valore iniziale di 100. Per far apparire il punteggio daremo un comando LOCATE e quindi, il comando PRINT.

```
PRINT USING "POSTA: # # # #";P
```

Questo nuovo comando dice al computer il **formato** in cui deve far apparire l'informazione. Il segno # significa "cifra". Questo comando dice al computer: fa' apparire la variabile P usando i quattro spazi vuoti che seguono sullo schermo "POSTA": Il gioco inizia con una posta di 100 e così la prima cosa che il computer visualizzerà all'inizio del gioco nella posizione 2,18 sarà:

```
POSTA:   100
         4 cifre
```

La riga 26 dice al computer di far apparire la nostra scommessa sullo schermo, proprio sotto la posta. Siccome anche la nostra scommessa cambierà ogni volta, questa riga usa la variabile B per indicare la scommessa. Il comando INPUT dice al computer di aspettare che il valore di B—la scommessa—venga introdotto dalla tastiera. Se scommettiamo 10 punti, verranno visualizzati nel modo seguente:



Dopo aver eseguito le righe 23 e 26, il computer passa alla sezione FOR-NEXT. A questo punto, immettiamo nel nostro programma le nuove righe di comando 23, 26, 103 e 106.

```

10 DIM A(2)
20 CLS
23 P=100:LOCATE 4,18:PRINT USING "POS
TA:####";P
26 LOCATE 0,20:INPUT "SCOMMESSA";B
30 FOR I=0 TO 2
40 A(I)=INT(RND(1)*4)+1
50 LOCATE I*3+12,9
60 ON A(I) GOTO 70,80,90,100
70 PRINT "♥":GOTO 103
80 PRINT "♠":GOTO 103 ← Cambia
90 PRINT "♦":GOTO 103
100 PRINT "♣"
103 K$=INKEY$
106 IF K$="A" THEN END ← Aggiungi
110 NEXT I
120 GOTO 30

```

Notate che dobbiamo cambiare il comando GOTO 110 delle righe 70, 80 e 90 e sostituirlo con GOTO 103. Che cosa succederebbe se ce ne dimenticassimo?

A questo punto, la macchinetta mangiasoldi è pronta ad accettare le nostre scommesse. L'unica cosa che non abbiamo ancora detto al computer è come cambiare la posta iniziale, vale a dire, come tenere il punteggio. Dovremo perciò aggiungere ancora qualche nuovo comando, che comincerà col numero di riga 130, alla fine del programma. Ma, prima, torniamo alla riga 106.

```
106 IF K$="A" THEN END
```

Quando noi premiamo il tasto **A** per fermare il movimento dei simboli, in realtà non vogliamo terminare il programma. Piuttosto, vogliamo che il computer ci dica qual'è la situazione del punteggio. Perciò cambiamo la riga 106 in questo modo:

```
106 IF K$="A" THEN GOTO 130
```

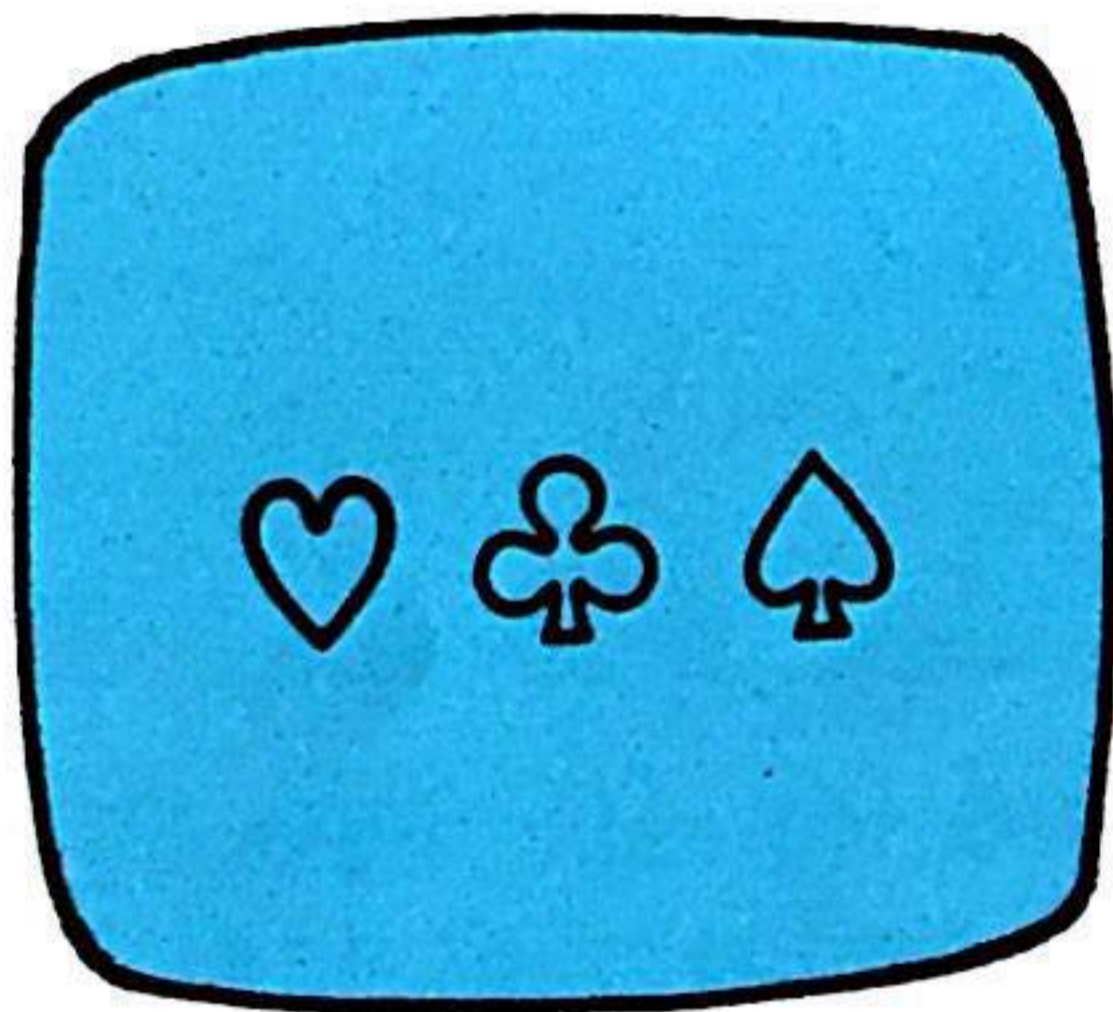

Quando vediamo i simboli sullo schermo, sappiamo se abbiamo vinto o perso. Se sono tutti e tre uguali, vinciamo il triplo di quello che abbiamo scommesso (se la nostra posta iniziale è di 100 e scommettiamo 10, nel caso vedessimo sullo schermo tre simboli uguali, il nostro nuovo punteggio sarebbe 130). Se due dei tre simboli sono uguali, vinciamo quello che scommettiamo (il nuovo punteggio sarebbe quindi 110). Se tutti e tre i simboli sono diversi, perdiamo il doppio di quello che abbiamo scommesso (il nuovo punteggio sarebbe 80). Se, per esempio, vediamo



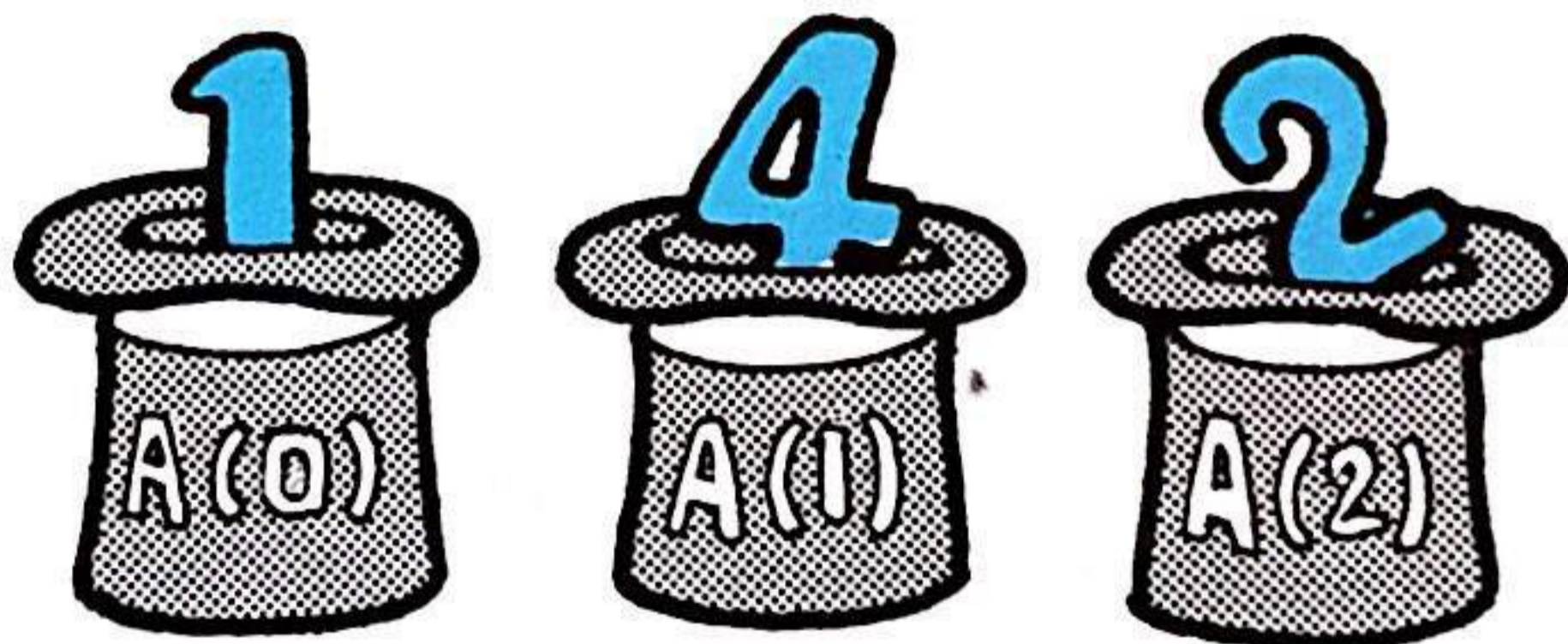
sappiamo che abbiamo perso qualche punto.

A differenza di noi, il computer non guarda i simboli sullo schermo. Esso usa le variabili contenute nella sua memoria. Al computer basterà controllare i valori delle variabili insieme A(0), A(1) e A(2), per sapere immediatamente se i loro valori sono uguali.

Questo è quello che vediamo



Questo è quello che c'è nella memoria del computer



Se sullo schermo viene visualizzato:



che cosa ci sarà nella memoria del computer?

Come possiamo vedere, non è molto difficile per il computer decidere se due, tre o nessuno dei simboli sono uguali. Ciò significa che anche i comandi che dobbiamo inserire nel programma per il calcolo dei punti non sono molto difficili.

```

130 IF A(0)=A(1) AND A(0)=A(2) THEN P
=P+B*3:GOTO 150
140 IF A(0)=A(1) OR A(1)=A(2) OR A(0)
=A(2) THEN P=P+B ELSE P=P-B*2
150 LOCATE 10,18:PRINT USING "####";P
160 GOTO 26

```

Questi nuovi comandi cominciano con la riga 130. Ciò significa che la parte del programma riguardante il punteggio verrà usata solo dopo che il tasto **A** sarà stato premuto e la macchinetta mangiasoldi si sarà fermata.


```
130 IF A(0)=A(1) AND A(0)=A(2) THEN P
=P+B*3:GOTO 150
```

Come possiamo notare, il comando IF—THEN contiene **due** condizioni collegate con **AND**. Se le condizioni sono **entrambe** vere— $A(0)=A(1)$ e $A(0)=A(2)$ —allora, tutte e tre le variabili insieme saranno uguali e i tre simboli che appariranno sullo schermo saranno uguali. Se ciò è vero

```
THEN P=P+B*3:GOTO 150
```

Questo significa che la nostra scommessa verrà moltiplicata per 3 e aggiunta alla posta P ; quindi, il computer andrà alla riga 150. Se scommettiamo 10 con una posta iniziale di 100, il nostro nuovo punteggio sarà 130.

Se però le tre variabili insieme non sono uguali, il computer non seguirà il comando THEN, ma passerà invece alla riga successiva del programma.

```
140 IF A(0)=A(1) OR A(1)=A(2) OR A(0)
=A(2) THEN P=P+B ELSE P=P-B*2
```

Su questa riga, troviamo un comando IF con tre condizioni collegato con **OR**. Questo significa che se **una** coppia qualsiasi di variabili è uguale (oppure, se due simboli sono uguali),

```
THEN P=P+B
```

aggiunge quello che abbiamo scommesso alla posta iniziale portando a 110 il nostro punteggio. Tuttavia, se nessuna delle tre condizioni è vera (tutti i simboli sono diversi tra loro), il computer andrà a

```
ELSE P=P-B*2
```

Abbiamo perso! La nostra scommessa viene moltiplicata per due e sottratta dalla posta iniziale—riducendo il nostro punteggio a 80.

Ora il computer conosce il punteggio e ce lo mostrerà al prossimo passaggio.

```
150 LOCATE 10,18:PRINT USING "####";P
```

Il passaggio seguente è piuttosto semplice da capire perché è quasi uguale al comando del formato PRINT che compare alla riga 23: prima il comando LOCATE, poi il comando PRINT USING con il formato di quattro cifre, infine il nome della variabile che sta prendendo un nuovo valore. Ogni volta che noi fermiamo la macchinetta mangiasoldi, il computer calcolerà il nuovo punteggio e lo farà apparire alla posizione 10,18.

Cosa succederà dopo? Il gioco continua, nel senso che dovremo fare un'altra scommessa. La riga 160, perciò, rimanda il computer alla riga 26, dove esso aspetta che noi gli comunichiamo la nostra nuova scommessa dalla tastiera per riprendere il gioco.

Immettiamo ora nel programma i nuovi comandi riguardanti il punteggio. Lo schermo dovrebbe avere questo aspetto:

```
10 DIM A(2)
20 CLS
23 P=100:LOCATE 4,18:PRINT USING "POS
TA:####";P
26 LOCATE 0,20:INPUT "SCOMMESSA";B
30 FOR I=0 TO 2
40 A(I)=INT(RND(1)*4)+1
50 LOCATE I*3+12,9
60 ON A(I) GOTO 70,80,90,100
70 PRINT "♥":GOTO 103
80 PRINT "♠":GOTO 103
90 PRINT "♦":GOTO 103
100 PRINT "♣"
103 K$=INKEY$
106 IF K$="A" THEN GOTO 130 ← Cambia
110 NEXT I
120 GOTO 30
130 IF A(0)=A(1) AND A(0)=A(2) THEN P
=P+B*3:GOTO 150
140 IF A(0)=A(1) OR A(1)=A(2) OR A(0)
=A(2) THEN P=P+B ELSE P=P-B*2
150 LOCATE 10,18:PRINT USING "####";P
160 GOTO 26
```

↑
Aggiungi

Aspettate un momento! Questo programma è troppo lungo e non ci sta sullo schermo. La prima parte scompare quando immettiamo il comando LIST. Per risolvere questo semplice problema, dobbiamo dire al computer **che cosa** listare, in questo modo:

```
LIST 10-100
```

Con questo comando, viene visualizzata la sezione che va dalla riga 10 alla 100. Per vedere l'ultima parte, immettiamo

LIST 110-

Ora, usando il comando LIST, controlliamo attentamente che non ci siano errori. Quando siamo sicuri che tutto il programma sia corretto, battiamo il comando RUN—e buon divertimento con la nuova macchinetta mangiasoldi!

GLI ULTIMI RITOCCHI

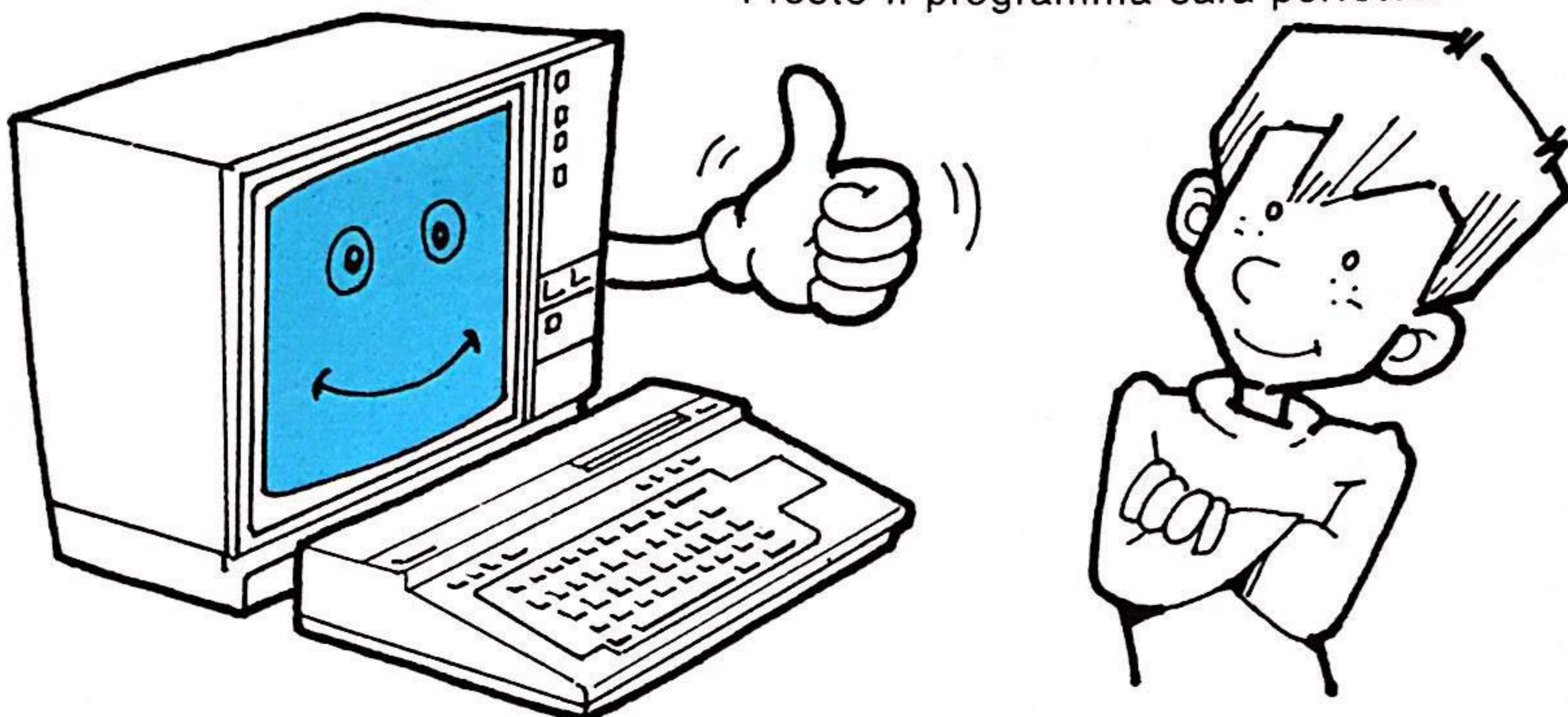
Il nostro computer Sony è così diventato una macchinetta mangiasoldi che funziona perfettamente. Sa fare alcune operazioni piuttosto difficili: richiedere una scommessa, cambiare i simboli delle tre finestrelle e calcolare il punteggio quando noi fermiamo il movimento dei simboli. Il programma che abbiamo realizzato per compiere tutte queste cose contiene solo 20 righe—non molte, se pensiamo che bisogna dire **ogni** volta al computer di visualizzare, di calcolare, di fermare, di aspettare la scommessa, e così via.

Tuttavia, manca ancora una cosa. All'inizio abbiamo deciso che si vince il gioco quando si raggiungono i 300 punti e lo si perde quando il punteggio scende a 0. A questo punto ormai, chissà di quanto avremo già superato i 300 punti, o chissà di quanto saremo scesi sotto lo 0, senza che il gioco sia stato né vinto, né perso.

Cerchiamo di rendere ancora migliore il nostro programma aggiungendo dei comandi per interrompere il gioco quando un certo punteggio viene raggiunto, in modo da poter vincere o perdere il gioco. Nello stesso tempo introdurremo qualche altro miglioramento che renderà il nostro programma veramente **perfetto**.

- Possiamo allargare un po' il nostro gioco rendendo possibile l'introduzione di un'altra, diversa scommessa—dopo aver premuto la barra di spazio.
- La barra di spazio, inoltre, sarebbe più comoda del tasto **A** per fermare il gioco, perché è più facile da battere.
- Quando facciamo la nuova scommessa, la vecchia è ancora visibile sullo schermo. Questo può farci fare confusione.

Presto il programma sarà perfetto!



Queste sono le modifiche che dovremmo inserire nel nostro programma.

```
10 DIM A(2)
20 CLS
23 P=100:LOCATE 4,18:PRINT USING "POS
TA:####";P
24 LOCATE 10,20:PRINT "      " ← Aggiungi
26 LOCATE 0,20:INPUT "SCOMMESSA";B
30 FOR I=0 TO 2
40 A(I)=INT(RND(1)*4)+1
50 LOCATE I*3+12,9
60 ON A(I) GOTO 70,80,90,100
70 PRINT "♥":GOTO 103
80 PRINT "♠":GOTO 103
90 PRINT "♦":GOTO 103
100 PRINT "♣"
103 K$=INKEY$
106 IF K$=" " THEN GOTO 130
110 NEXT I ↑ Cambia
120 GOTO 30
130 IF A(0)=A(1) AND A(0)=A(2) THEN P
=P+B*3:GOTO 150
140 IF A(0)=A(1) OR A(1)=A(2) OR A(0)
=A(2) THEN P=P+B ELSE P=P-B*2
150 LOCATE 10,18:PRINT USING "####";P
```



```

152 GOTO 170
154 K$=INKEY$
156 IF K$=" " THEN GOTO 24
160 GOTO 154
170 IF P<300 AND P>0 THEN GOTO 154
180 IF P>=300 THEN LOCATE 3,5:PRINT "
HAI VINTO!"
190 IF P<=0 THEN LOCATE 3,5:PRINT "HA
I PERSO!"
200 END

```

← Aggiungi

← Cambia

↑ Aggiungi

Avete capito? Prima di leggere la spiegazione, pensateci alcuni minuti per vedere se lo capite da soli.

Dalla riga 170 comincia una serie di comandi che dicono al computer di comunicarci le vincite o le perdite.

Se il punteggio è minore di 300 e (AND) maggiore di 0, il gioco continua e il computer ritorna al programma. Se il punteggio è 300 o più di 300, il gioco è finito e il computer ce lo comunica facendo apparire 'HAI VINTO!' vicino alla parte superiore dello schermo. Se il punteggio è meno di 0, sullo schermo appare 'HAI PERSO!'. Sia che si perda, sia che si vinca, il computer va comunque alla riga 200, dove il comando END pone fine al gioco.

La riga 24 cancella la scommessa precedente all'inizio di ogni giro del gioco. Essa porta il cursore nella posizione in cui la scommessa è visualizzata e stampa sei spazi vuoti al posto della scommessa stessa.

Il cambiamento introdotto dalla riga 106 è molto semplice. Per fermare il gioco, dobbiamo premere la barra di spazio invece del tasto **A**. Ricordate che il computer legge gli **spazi** (" ") come se fossero **caratteri**.

Le righe 160, 154 e 156 significano che il gioco riprenderà quando premeremo la barra di spazio.

Questi sono i quattro cambiamenti che volevamo fare: i comandi per vincere/perdere il gioco e i tre piccoli miglioramenti.

La riga 152 non modifica il gioco ma è stata resa necessaria dalle modifiche introdotte nel programma. Questa riga, che viene subito dopo che il nuovo punteggio è stato visualizzato, manda il computer più avanti, alla sezione dei comandi per vincere/perdere che comincia dalla riga 170. Se non abbiamo né vinto né perso, la riga 170 rimanda il programma alla riga 154 e il gioco continua fino a quando non premiamo la barra di spazio.

PER LEGGERE PIÙ FACILMENTE IL PROGRAMMA —

Quando abbiamo cominciato a scrivere il programma della macchinetta mangiasoldi, abbiamo numerato le righe per decine: 10, 20, 30, 40 ... Poi, per rendere più completo il nostro programma, abbiamo usato alcune delle righe "vuote" per aggiungere i nuovi comandi. Di conseguenza, le righe del nostro programma sono ora numerate irregolarmente: 90, 100, 103, 106, 110, 120. Perché il programma risulti più facile da leggere, possiamo **rinumerare** le righe.

Introduciamo questo comando:

RENUM

L'ordine delle righe rimane lo stesso, ma i numeri di riga vengono tutti ordinati a decine, come possiamo verificare se diamo il comando LIST.

In questo modo, i numeri di riga sono effettivamente più facili da leggere. Ma cosa sarà successo ai nostri comandi GOTO che contenevano tutti dei numeri di riga? Non c'è problema! Proviamo a guardare, per esempio, la riga 100 del nostro nuovo programma:

```
100 PRINT "♥":GOTO 140
```

Questo comando GOTO—GOTO 140—era GOTO 103 prima che dessimo il comando RENUM. La riga 103 del vecchio programma è diventata la riga 140 e GOTO 103 è diventato GOTO 140. Ciò significa che il nostro computer Sony è davvero sveglio e il nostro gioco, quando faremo girare di nuovo il programma delle macchinette mangiasoldi, sarà ancora tale e quale.

METTIAMO IL TITOLO AL PROGRAMMA

Supponiamo di voler mostrare il programma ad un amico o supponiamo di voler rileggere il programma alcuni mesi dopo averlo scritto. Quando la riga 20 o la 30 del programma, scritte in BASIC, appaiono sullo schermo, può essere difficile, per una persona che non è pratica di linguaggio BASIC, capire le loro funzioni. Per risolvere questo problema, il BASIC vi lascia inserire in mezzo al programma delle "osservazioni", in questo modo:

```
5 REM *GIOCO DELLA MACCHINETTA MANGIA
SOLDI
7 REM
10 DIM A(2)
20 CLS
30 P=100:LOCATE 4,18:PRINT USING "POS
TA:####";P
40 LOCATE 10,20:PRINT "      "
50 LOCATE 0,20:INPUT "SCOMMESSA";B
55 /
60 / ** LA MACCHINETTA MANGIASOLDI CO
MINCIA A GIRARE
65 FOR I=0 TO 2          — Precedentemente riga 60
70 A(I)=INT(RND(1)*4)+1
80 LOCATE I*3+12,9
90 ON A(I) GOTO 100,110,120,130
100 PRINT "♥":GOTO 140
110 PRINT "♠":GOTO 140
120 PRINT "♦":GOTO 140
130 PRINT "♣"
140 K$=INKEY$
150 IF K$=" " THEN GOTO 180
160 NEXT I
170 GOTO 60
175 /
180 / *** CALCOLA LA POSTA ***
182 /
185 IF A(0)=A(1) AND A(0)=A(2) THEN P
=P+B*3:GOTO 200          — Precedentemente riga 180
190 IF A(0)=A(1) OR A(1)=A(2) OR A(0)
=A(2) THEN P=P+B ELSE P=P-B*2
200 LOCATE 10,18:PRINT USING "####";P
```



```

210 GOTO 250
220 K$=INKEY$
230 IF K$=" " THEN GOTO 40
240 GOTO 220
245 /
250 / *** SI VINCE O SI PERDE ***
252 /
255 IF P<300 AND P>0 THEN GOTO 220 ← Precedentemente riga 250
260 IF P>=300 THEN LOCATE 3,5:PRINT "
HAI VINTO!"
270 IF P<=0 THEN LOCATE 3,5:PRINT "HA
I PERSO!"
280 END

```

Le nostre "osservazioni" riguardo al programma diventano il **titolo** e i **sottotitoli** delle sue diverse parti. Tali osservazioni allungano il programma, ma lo rendono molto più facile da leggere e da capire. Senza contare che non cambiano assolutamente il programma, perché il computer non legge le righe che contengono solo delle osservazioni—esse servono solo alla persona che usa il programma.

Diamo un'occhiata ad alcune di queste nuove righe di osservazione:

```

5 REM *GIOCO DELLA MACCHINETTA MANGIA
SOLDI
7 REM

```

Il comando REM dice al computer di **ignorare** i caratteri che seguono il comando stesso. Si può inserire qualsiasi cosa dopo questo comando ed esso non cambierà minimamente il programma. Sulla riga 7, però, non c'è scritto niente dopo il comando REM. Non è stato scritto niente apposta perché si formi uno spazio vuoto attorno al titolo—proprio come sulla prima pagina di un libro. Lo spazio vuoto e le stelline, prima e dopo il titolo, ci permetteranno di individuare facilmente il titolo in mezzo a tutte quelle righe di comandi BASIC.

Passiamo ora a considerare le righe 55, 60 e 62. Come possiamo vedere, c'è un modo più veloce per dare il comando REM: con il segno ' (apostrofo). Queste righe, ovviamente, non fanno parte del programma. Poiché iniziano con ' il computer non le considererà delle righe di comando. Il sottotitolo, le stelline e lo spazio vuoto attorno al titolo sono per noi, non per il computer.

Il computer **vede** i numeri di riga perché essi vengono prima del comando ' (o REM). Guardiamo, per esempio, la riga 150:

```
150 IF K#=" " THEN GOTO 180
```

La riga 180, invece, è una riga di commento:

```
180 / *** CALCOLA LA POSTA ***
```

Poiché non trova nessun comando alla riga 180, il computer passa ad elaborare la riga seguente, ma non essendoci nessun comando nemmeno su quella riga, la 182, il computer va alla 185. Il **lettore** del programma, però,—noi o i nostri amici—passa dalla riga 150 alla riga 180 e capisce immediatamente che questa è la parte dove il computer calcola i punti che il giocatore possiede.

NE ABBIAMO FATTA DI STRADA! ---

Complimenti! Dopo aver fatto gli esercizi e i giochi contenuti in questo libro, potete dire di avere una conoscenza pratica di una nuova lingua—il BASIC. Forse sono passate solo alcune ore, o un paio di giorni da quando abbiamo cominciato a usare insieme dei comandi molto semplici come PRINT 3 + 5. Ora capite e sapete usare righe di comando complicate come $A(I)=INT(RND(1) * 4) + 1$ e PRINT USING “# # # #”;P. Questo computer Sony vi servirà per anni, offrendovi divertimento e istruzione ora che sapete “parlare la sua lingua”.

Man mano che vi eserciterete a scrivere e ad usare i programmi, acquisite una sempre maggiore abilità con i numeri, i comandi, le funzioni e i giochi. È solo questione di tempo e presto sarete degli **esperti**. Giocherete in famiglia o con gli amici e passare il tempo non sarà certo un problema col computer Sony.

Potete continuare ad esercitarvi nell'uso dei comandi ed impararne degli altri che sono illustrati nella parte seguente di questo libro. Dopo di che, sarete pronti ad usare il **Manuale di riferimento per programmazione MSX-BASIC** per scoprire qual è la funzione di tutti gli altri comandi BASIC che vi sono spiegati. Così potrete cominciare a sperimentare col vostro computer, inventare colori, grafici, suoni e nuovi giochi.

Quando volete creare un **vostro programma personale**, spendete qualche minuto prima per prepararlo, cercando di rappresentarlo magari con un diagramma di flusso. Quando sapete di quali **funzioni** e di quali **comandi** avrete bisogno, usate l'indice che si trova in fondo a questo libro per trovarli e rivedetene le spiegazioni. Pensate attentamente e quindi, scrivete qualche comando e provate ad eseguirlo. Il vostro computer Sony vi aiuterà sempre in due modi: farà sempre **esattamente** quello che gli direte di fare, e dimenticherà subito i vostri errori quando immetterete il comando NEW.

Parlate dei vostri programmi con i vostri amici. Mostrate loro quello che potete far col vostro computer. Se la pensano diversamente da voi, com'è probabile, vi divertirte di più. Se anche loro possiedono un computer, potete scambiarsi i programmi, usando un registratore o scrivendo i programmi su carta.

Ricordate, il modo migliore per divertirvi con il vostro computer Sony è provare sempre qualcosa di nuovo e stare a vedere quello che succede. Non c'è limite alle cose che si possono dire al computer con il linguaggio BASIC. Siete pronti a partire per questo meraviglioso viaggio nella vostra fantasia? Buona fortuna!

ESERCITIAMOCI CON I COMANDI BASIC

PRINT

```
10 PRINT "PAOLO"  
20 PRINT "E"  
30 PRINT "MARIA"  
RUN  
PAOLO  
E  
MARIA
```

Vi ricordate questo semplice programma? Cambiamolo un po'.

```
10 PRINT "PAOLO";  
20 PRINT "E";  
30 PRINT "MARIA"  
RUN  
PAOLOEMARIA
```

Aggiungi

Con il segno ; (punto e virgola) dopo "PAOLO" e dopo "E", il computer fa apparire le tre parole tutte insieme sulla stessa riga. In effetti, i comandi di un programma potrebbero essere tutti sistemati su una sola riga:

```
10 PRINT "PAOLO";"E";"MARIA"  
RUN  
PAOLOEMARIA
```

Importante

Ma è difficile leggere tre parole se non ci sono spazi che le dividono, perciò proviamo qualcos'altro.

```
10 PRINT "PAOLO";  
20 PRINT "E";  
30 PRINT "MARIA"  
RUN  
PAOLO      E  
MARIA      |  
            | 14 caratteri
```

Importante

Quando usiamo una , (virgola) invece di un ; (punto e virgola), le prime lettere di ciascuna parola vengono stampate a una distanza di 14 spazi l'una dall'altra. Lo stesso succederà se scriviamo il comando su una sola riga.

```
10 PRINT "PAOLO" , "E" , "MARIA"  
RUN  
PAOLO           E           Importante  
MARIA           |           |  
                14 caratteri
```

Proviamo ora a scrivere questo comando battendo uno spazio tra ogni parola e le seconde virgolette.

```
10 PRINT "PAOLO " ; "E " ; "MARIA"  
RUN  
PAOLO E MARIA
```

E ora passiamo in rassegna i diversi modi in cui possono essere stampati i caratteri numerici.

```
10 PRINT "3+5=" ; ← Fa' apparire i caratteri  
20 PRINT 3+5      ← Esegue il calcolo e fa' apparire la risposta  
RUN  
3+5= 8
```

La riga 10 è un comando per far apparire i caratteri, mentre la riga 20 è un comando per l'esecuzione del calcolo. Queste due diverse funzioni sono **collegate** dal ; posto alla fine della riga 10. Il punto e virgola ha detto al computer di stampare le due funzioni insieme su una riga, nello stesso modo in cui il problema verrebbe scritto su un foglio di carta (Nella risposta c'è uno spazio vuoto davanti al numero 8. Esso viene usato per scrivere il segno meno (—) nel caso la risposta sia un **numero negativo** inferiore a zero—per esempio, $3 - 5 = -2$).

Usiamo i comandi INPUT e PRINT nello stesso semplice programma.


```

10 INPUT A
20 INPUT B
30 PRINT A,B,A+B
RUN
? 3      ← Che valore? Batti 3
? 5      ← Che valore? Batti 5

```

Valore di A Valore di B Valore di A+B

Il comando INPUT dice al computer di **chiederci** quale valore dare alla variabile e di aspettare fino a che noi non introduciamo quel valore mediante la tastiera. Dopo che è stato battuto il tasto **RETURN**, il computer passa al comando successivo. La riga 30 fa apparire sullo schermo i tre valori.

Questo è un modo diverso, più chiaro di eseguire le medesime funzioni.

```

10 INPUT "A=" ;A
20 INPUT "B=" ;B
30 C=A+B
40 PRINT "A+B=" ;C
RUN
A=? 3
B=? 5
A+B= 8 ← Risultato della riga 40

```

La riga 10 e la riga 20 dicono al computer quale domanda visualizzare sullo schermo, invece di usare il solo ? (punto interrogativo). La riga 30 crea una nuova variabile, C, e le assegna il valore di A+B. La riga 40 dice al computer di stampare la risposta e il problema, cominciando con A+B. Le tre righe che vengono dopo RUN sono molto più facili da capire dei tre numeri che apparivano da soli alla fine dell'esempio precedente.

Quello che viene visualizzato sullo schermo è talvolta più facile da leggere se le parole, o gruppi di parole, sono separate da **righe vuote**.


```
10 PRINT "PAOLO"  
20 PRINT  
30 PRINT "E"  
40 PRINT  
50 PRINT "MARIA"
```

RUN

PAOLO

Una riga vuota nel mezzo

E

Una riga vuota nel mezzo

MARIA

Se si scrive solo il comando PRINT su una riga (come nelle righe di comando 20 e 40), viene visualizzata sullo schermo una riga vuota.

INPUT

```
10 INPUT "A è ";A  
20 INPUT "B è ";B  
30 PRINT "A+B=";A+B  
40 PRINT "A-B=";A-B
```

RUN

A è ? 15

B è ? 3

A+B= 18

A-B= 12

Questo è altro programma che combina i comandi INPUT e PRINT. Non è difficile capire che si dice al computer di **chiedere** i valori di A e B e di **visualizzare** sullo schermo due calcoli. Proviamo ora a combinare due comandi INPUT facendo qualche altro calcolo.


```

10 INPUT "A e B sono ";A,B
20 PRINT "A=";A,"B=";B
30 PRINT
40 PRINT "A*B=";A*B
50 PRINT "A/B=";A/B
RUN
A e B sono ? 15,3
A= 15          B= 3

A*B= 45
A/B= 5

```

Nella riga 10 si possono immettere due valori per due diverse variabili,—facendo però attenzione alla , (virgola) che separa i due valori (in questo caso, 15 e 3), perché è **molto** importante. Se il computer facesse apparire 153 invece di 15,3, sarebbe difficile capire che cosa significa. (Anche la punteggiatura del comando PRINT alla riga 20 è molto importante. Sapete cosa significano la , e il ; ?).

Ora aggiungeremo una **variabile di stringa** al nostro comando INPUT.

```

10 INPUT "Nome";N$
20 PRINT N$;" è in gamba!"
RUN
Nome? Paolo
Paolo è in gamba!

```

Quando il nome della variabile termina con il segno \$, la variabile "assume il valore di" una lettera o di una parola. Ecco un programma che usa sia una variabile alfabetica che una variabile numerica.

```

10 INPUT "Nome";N$
20 INPUT "Età";Y
30 PRINT N$;" ha ";Y;" anni."
RUN
Nome? Paolo
Età? 10
Paolo ha 10 anni.

```


FOR—NEXT

```
10 CLS
20 FOR I=0 TO 36
30 LOCATE I,10
40 PRINT "$"
50 NEXT I
```



```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

In questo programma, la funzione di ripetizione (**ciclo**) del comando FOR—NEXT fa sì che il computer visualizzi ripetutamente sullo schermo il segno \$ dalla posizione 0,10 alla posizione 36,10.

Qui di seguito vediamo lo stesso programma al quale è stata però aggiunta una nuova istruzione.

```
10 CLS
20 FOR I=0 TO 36 STEP 3
30 LOCATE I,10
40 PRINT "$"
50 NEXT I
```



```
$ $ $ $ $ $ $ $ $ $ $ $ $ $ $
```

Che cosa significa **STEP 3** alla riga 20? Come possiamo vedere, questo comando ha fatto saltare al cursore **tre passi** dopo ogni segno \$ (STEP=passo). In altre parole, il valore di I in questo programma cambia in passi di 3 (di tre unità per volta), così che il segno \$ appare alle posizioni 0,10, 3,10 e 6,10...—e non alle posizioni 0,10, 1,10 e 2,10... I segni \$ coprono un'area dello schermo pressoché uguale a quella del programma precedente, vale a dire da 0,10 a 36,10.

Aggiungendo **STEP e un numero** al comando FOR, pertanto, possiamo **far cambiare la variabile in passi** e non in unità. Nell'ultimo esempio che abbiamo visto, la variabile era la posizione la quale, perciò, è andata cambiando in passi. Il programma seguente usa l'istruzione STEP per cambiare il modo di contare del computer.


```

10 FOR I=50 TO 0 STEP -5
20 PRINT I,
30 NEXT I
RUN
50          45
40          35
30          25
20          15
10          5
0

```

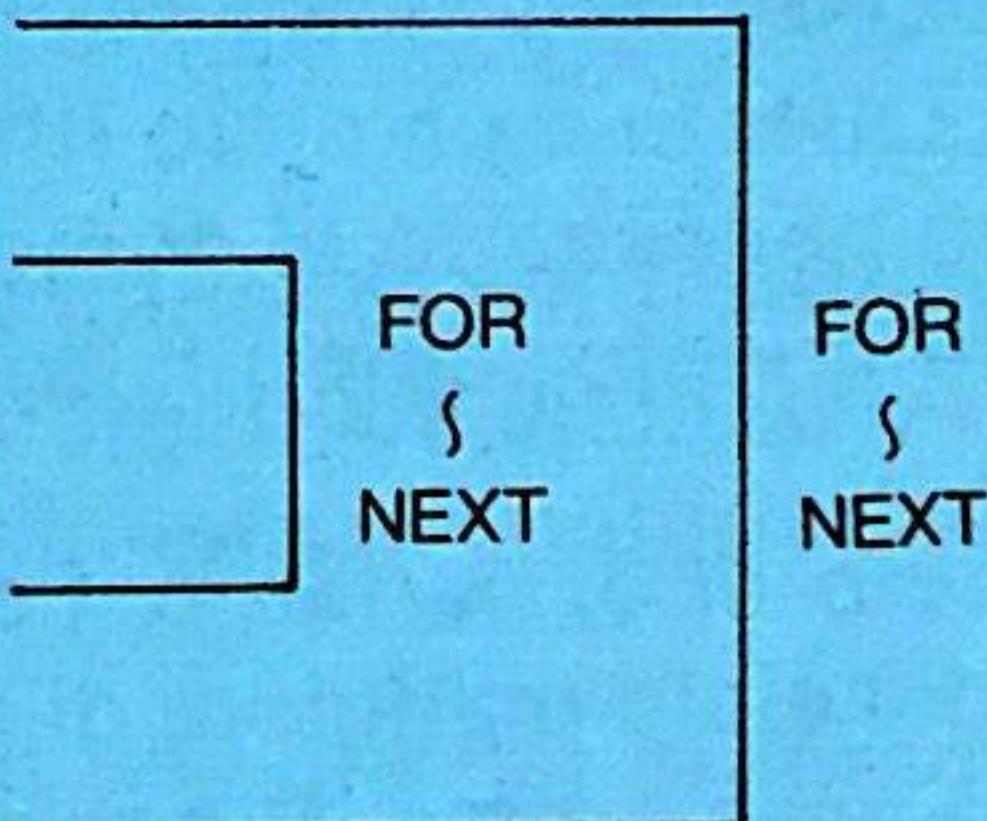
I=50 TO 0 dice al computer di contare come in ogni ciclo di ripetizione FOR—NEXT. I può prendere 51 valori tra 50 e 0. Tuttavia, il comando **STEP—5** diminuisce il valore di I in passi di -5, così che I può assumere solo 11 valori diversi.

FOR—FOR—NEXT—NEXT: Una ripetizione all'interno di una ripetizione

```

10 FOR I=0 TO 2
20 PRINT "I=";I
30 FOR J=0 TO 4
40 PRINT "J=";J;
50 NEXT J
60 PRINT
70 NEXT I

```



```

RUN
I= 0
J= 0  J= 1  J= 2  J= 3  J= 4
I= 1
J= 0  J= 1  J= 2  J= 3  J= 4
I= 2
J= 0  J= 1  J= 2  J= 3  J= 4

```

Questo programma ci mostra come inserire una ripetizione FOR—NEXT all'interno di un'altra ripetizione FOR—NEXT più ampia. La variabile I è controllata dalle prime e dalle ultime righe del programma, mentre la variabile J è controllata dalle righe 30 e 50. Ne risulta che, per ogni valore di I, il computer compie quattro piccole ripetizioni per trovare tutti i valori di J. I varia da 0 a 2 e, con ogni variazione di I, J varia da 0 a 4.

Vediamo ora un altro modo di usare una ripetizione.

```
10 INPUT N,S
20 CLS
30 FOR I=0 TO N STEP S
40 LOCATE 5,10:PRINT "I=";
50 PRINT USING "####";I
60 FOR J=0 TO 500
70 NEXT J
80 NEXT I
```

Questo programma usa i comandi INPUT, FOR—NEXT e STEP per far cambiare la variabile I in cicli di ripetizione da 0 a N, in passi di S. È ovvio che assegneremo valori a N e a S usando la tastiera.

Quale sarà la funzione della variabile J e della breve ripetizione FOR—NEXT alle righe 60 e 70? J non viene mai stampata. In effetti, questa variabile non viene usata per niente, ma il computer troverà per J 501 valori e glieli assegnerà uno dopo l'altro prima di passare a calcolare il valore di I e a stamparla. Questo è un ciclo **fittizio**. Per eseguirlo ci vuole un po' di tempo—il tempo necessario al computer per ripetere 501 cicli FOR—NEXT—ma durante questo tempo il computer non fa nient'altro.

La funzione del ciclo di J, pertanto, consiste nel porre un **intervallo di tempo** all'interno del ciclo maggiore di I. Come possiamo vedere quando eseguiamo questo programma, il computer fa una piccola pausa dopo aver visualizzato ogni nuovo valore di J. Ora potete verificare quanto tempo ci mette il nostro computer Sony ad eseguire 501 semplici operazioni (abbastanza poco, diremmo!), e sapete anche come usare un ciclo fittizio per inserire un po' di tempo extra nei vostri programmi. Sapreste scrivere un ciclo fittizio che tenga occupato il computer per un tempo ancora più lungo?

IF—THEN e IF—THEN—ELSE

Ora vedremo un altro gioco a indoviniello in cui dovremo cercare di indovinare il numero che il computer sceglierà. Questo gioco è un po' diverso da quello in cui dovevamo indovinare un numero e da quello della macchinetta mangiasoldi che abbiamo programmato precedentemente in questo libro. A differenza di quei giochi, nei quali era solo questione di fortuna, stavolta possiamo far uso di una certa **tecnica** per indovinare più rapidamente il numero corretto. Se siamo abili, possiamo indovinare il numero con sei o sette tentativi. Se tiriamo a indovinare, però, potrebbero volercene di più.

```
10 X=INT(RND(1)*100)+1
20 INPUT "Prova a indovinare";A
30 IF A=X THEN GOTO 70
40 IF A>X THEN PRINT "PIÙ PICCOLO"
50 IF A<X THEN PRINT "PIÙ GRANDE"
60 GOTO 20
70 PRINT "Azzecato!"
```

Le righe 30, 40 e 50 dicono al computer di confrontare il numero che noi cerchiamo di indovinare con il valore di X e di darci un suggerimento che possa aiutarci ad indovinare il numero al tentativo seguente.

Riuscite a capire che cosa c'è di diverso nel programma che segue?

```
10 X=INT(RND(1)*100)+1
20 INPUT "Prova a indovinare";A
30 IF A=X THEN 70
40 IF A>X THEN PRINT "PIÙ PICCOLO"
50 IF A<X THEN PRINT "PIÙ GRANDE"
60 GOTO 20
70 PRINT "Azzecato!"
```

Questo programma è effettivamente uguale a quello precedente, anche se la riga 30 è stata cambiata. IF—THEN ha lo stesso significato di IF—THEN—GOTO. Il programma sarebbe lo stesso anche se la riga 30 fosse: 30 IF A=X GOTO 70. In altre parole, IF—GOTO ha lo stesso significato di IF—THEN—GOTO. Pertanto, è possibile eliminare o THEN o GOTO dall'istruzione (ma non entrambi!).

Nel programma seguente, proveremo a combinare due righe in una:

```
10 X=INT(RND(1)*100)+1
20 INPUT "Prova a indovinare";A
30 IF A=X THEN 70 ELSE
    IF A>X THEN PRINT "PIÙ PICCOLO"
50 IF A<X THEN PRINT "PIÙ GRANDE"
60 GOTO 20
70 PRINT "Azzecato!"
```

Come possiamo vedere, i due comandi IF—THEN contenuti nelle righe 30 e 40 sono stati combinati per formare un solo comando, IF—THEN—ELSE. Non c'è più la riga 40, ma il significato è il medesimo. Se questa operazione è possibile, allora dovrebbe essere possibile anche combinare le righe 30 e 50 di questo programma per formarne una sola.

```
10 X=INT(RND(1)*100)+1
20 INPUT "Prova a indovinare";A
30 IF A=X THEN 70 ELSE
    IF A>X THEN PRINT "PIÙ PICCOLO"
    ELSE PRINT "PIÙ GRANDE"
60 GOTO 20
70 PRINT "Azzecato!"
```

Possiamo controllare per vedere se questi quattro programmi sono effettivamente uguali, eseguendoli uno per uno sul computer.

DIM (Variabili insieme) ---

Quando abbiamo usato le variabili insieme nel gioco della macchinetta mangiasoldi, abbiamo usato il comando

```
DIM A (2)
```

per creare tre variabili:

```
A (0)    A (1)    A (2)
```


Quindi, abbiamo cambiato i numeri contenuti tra parentesi in una variabile, chiamata (I), e abbiamo detto al computer che I=0 TO 2 (TO significa 'sino a...'). Dopo questa operazione, avevamo ancora tre variabili insieme il cui valore veniva variato dal computer in un ciclo di ripetizione FOR—NEXT.

Le variabili insieme possono contenere più di un carattere all'interno delle parentesi. Per esempio, se scriviamo

DIM A (3,4)

avremo 20 variabili:

A (0, 0)	A (1, 0)	A (2, 0)	A (3, 0)
A (0, 1)	A (1, 1)	A (2, 1)	A (3, 1)
A (0, 2)	A (1, 2)	A (2, 2)	A (3, 2)
A (0, 3)	A (1, 3)	A (2, 3)	A (3, 3)
A (0, 4)	A (1, 4)	A (2, 4)	A (3, 4)

Capite queste 20 diverse variabili? Il primo numero contenuto tra parentesi può prendere quattro valori (da 0 a 3) e il secondo numero può avere cinque valori (da 0 a 4)—4 per 5 fa 20 (Quante variabili avremmo se scrivessimo DIM A(9,9)?).

Per illustrare le 20 variabili, non le abbiamo semplicemente elencate una dopo l'altra, ma le abbiamo disposte in una **tabella** (Provate a pensare a un insieme bidimensionale, con il primo valore che cambia da sinistra a destra e il secondo valore che cambia dall'alto verso il basso). Come possiamo constatare, le variabili insieme sono molto utili per i programmi che scrivono grafici o tabelle. Per usare queste variabili insieme bidimensionali in un programma, dovremmo sostituire i numeri all'interno delle parentesi con delle variabili, come in A(I,J).

Ecco l'esempio di una tabella vera:

	Italiano	Inglese	Matematica	Totale
1 trimestre	68	88	70	226
2 trimestre	73	53	91	217
3 trimestre	92	98	82	272
Totale	233	239	243	715
Media	77	79	81	238

Questa tabella illustra i voti di uno studente in tre materie per tre trimestri. Contando i totali e le medie, nella tabella ci sono esattamente 20 numeri. Ciò significa che possiamo usare il comando DIM (3,4) per creare un numero di variabili sufficienti per questa tabella. Ora, proviamo a pensare a come potremmo usare dei programmi per assegnare dei valori reali alle nostre variabili insieme A (I,J). Eccovi una possibilità:

```
100 FOR J=0 TO 2
110 INPUT A(0,J)
120 NEXT J
```

Queste tre righe sono per i punteggi in italiano del nostro studente. I tre voti diventano tre variabili: A (0,0), A (0,1) and A (0,2). Ogni volta che il computer arriva alla riga 110 del ciclo FOR—NEXT, ci chiederà di immettere i valori (68, 73, 92).

Poi, per calcolare il totale dei tre punteggi in italiano immettiamo:

```
200 T=0
210 FOR J=0 TO 2
220 T=T+A(0,J)
230 NEXT J
240 A(0,3)=T
```

..... T è la variabile per il totale del
punteggio in italiano

Procedendo nello stesso modo, possiamo immettere il punteggio in inglese del primo trimestre nella variabile A (1,0), e il punteggio in matematica del primo trimestre nella variabile (2,0). Per immettere il totale del punteggio nelle tre materie durante il primo trimestre nella variabile A (3,0), possiamo usare il seguente programma.

```
300 T1=0
310 FOR I=0 TO 2
320 T1=T1+A(I,0)
330 NEXT I
340 A(3,0)=T1
```

..... T1 è la variabile per il totale del
primo trimestre

Per programmare gli altri punteggi, i totali e le medie, possiamo usare cicli FOR—NEXT analoghi con variabili insieme che abbiano la forma di A (I,J). Il programma completo avrà perciò l'aspetto seguente:


```

10 / *PROGRAMMA PER LA TABELLA DEI PUNTEGGI
20 /
30 DIM A(3,4)
40 CLS
50 /
60 / * IMMETTI I PUNTEGGI *
70 FOR I=0 TO 2
80 FOR J=0 TO 2
90 ON I+1 GOTO 100,120,140
100 LOCATE 0,J:PRINT "Italiano,Trimestre ";J+1;
110 INPUT A(0,J):GOTO 160
120 LOCATE 0,J+3:PRINT "Inglese,Trimestre ";J+1;
130 INPUT A(1,J):GOTO 160
140 LOCATE 0,J+6:PRINT "Matematica,Trimestre ";J+1;
150 INPUT A(2,J)
160 NEXT J
170 NEXT I
180 /
190 / * CALCOLA I TOTALI E LE MEDIE *
200 FOR J=0 TO 2
210 T=0
220 FOR I=0 TO 2
230 T=T+A(I,J)
240 NEXT I
250 A(3,J)=T
260 NEXT J
270 FOR I=0 TO 3
280 T=0
290 FOR J=0 TO 2
300 T=T+A(I,J)
310 NEXT J
320 A(I,3)=T
330 A(I,4)=INT(A(I,3)/3)
340 NEXT I
350 /
360 / *** FA'UNA TABELLA ***
370 CLS

```



```

380 LOCATE 5,0
390 PRINT "ITAL ING    MAT    TOTALI"
400 FOR S=1 TO 3
410 LOCATE 2,S+1:PRINT S
420 NEXT S
430 LOCATE 1,5:PRINT "TTL"
440 LOCATE 1,6:PRINT "MED"
450 FOR I=0 TO 3
460 FOR J=0 TO 4
470 LOCATE I*6+4,J+2
480 PRINT A(I,J)
490 NEXT J
500 NEXT I

```

Per facilitare la preparazione del programma, lo abbiamo diviso in tre sezioni—Immetti i punteggi, Calcola i totali e le medie e Fa' una tabella. Inoltre, se annotiamo su un foglio di carta i punti in cui viene immessa ogni variabile, ci risulterà più facile ricordare come funziona l'intero programma.

Le variabili insieme, naturalmente, non vengono usate solo per costruire "macchinette mangiasoldi" o per realizzare delle tabelle. Inoltre, come forse avrete già indovinato, le variabili contenute tra parentesi possono essere più di due. Possiamo avere, per esempio, A (P,Q,R) o B (X,Y,Z,XY,XZ,YZ) ... o qualsiasi altra combinazione, fino a un massimo di 255 variabili diverse! I professionisti di computer, spesso, usano le variabili insieme per creare video-giochi, per calcolare le entrate e le uscite di grosse compagnie finanziarie o per eseguire altre complicate operazioni. Il vostro computer Sony può usare le variabili insieme per fare tante cose diverse e ve ne accorgete presto, programmando con i vostri amici, o esercitandovi con altri libri. Avendo già usato le variabili insieme in due diversi programmi, avete già una certa esperienza.

Pertanto, continuate a fare esercizio e buona fortuna!

INDICE ANALITICO

- C**
Caratteri 38
Ciclo 39
Ciclo (ripetizione)
condizionale 69
Ciclo fittizio 110
CLS 65
COLOR..... 46
Comandi..... 8
CSAVE..... 54
CTRL, Tasto 8
- D**
DIM 79, 112
Due punti (:)..... 66
- E**
ELSE (IF—THEN—ELSE). 64, 111
Errori..... 32
- F**
Formato..... 90
FOR—NEXT..... 69, 108
- G**
GOTO 38
Grafici..... 37
- I**
IF—THEN..... 61, 111
IF—THEN—GOTO 61
Immissioni 8
Indirizzo..... 65
INKEY 89
INPUT 60, 106
INT (intero)..... 44
- L**
LET..... 19
LINE..... 49
Lineetta (—)..... 49
LIST..... 35
LOCATE 68
- M**
Messaggio d'errore 9
MOD (modulo)..... 72
- N**
NEW..... 35
Nome di file..... 54
Numeri casuali..... 43
Numero di riga..... 35
Numero negativo..... 105
- O**
ON—GOTO..... 83
Ordine di esecuzione 28
Osservazioni 101
- P**
Parentesi ()..... 42
PRINT 14, 17, 103
PRINT USING 90
Programmazione..... 28
PSET 38
Punteggiatura..... 42
Punto e virgola (;)..... 103
- R**
REM 100
RENUM..... 98
RETURN, Tasto..... 9
Rinumerare 98
RND (1) 43
RUN 34
- S**
SCREEN 38
Segno del dollaro \$..... 86
SHIFT, Tasto 14
Sottotitoli 100
Spazio..... 10
STEP 108
STOP, Tasto 8
Stringa..... 85

T

Tabella dei colori	12
Tabelle.....	113
Tastiera.....	6
Titoli.....	100
TO.....	71

U

USING (PRINT USING).....	90
--------------------------	----

V

Valore iniziale.....	70
Variabili	20
Variabili di stringa	25, 68, 85
Variabili insieme	77-78, 112
Virgola (,).....	42
Virgola decimale (.)	43
Virgolette (").....	17

W

WIDTH.....	10
------------	----

