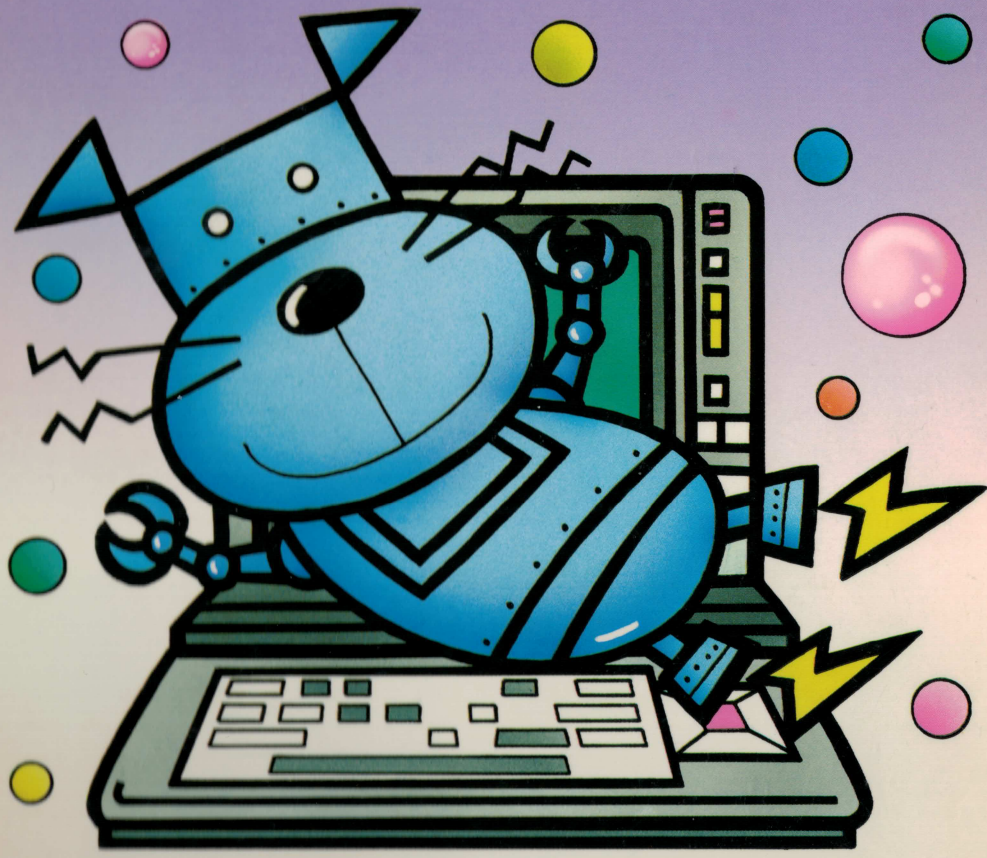
Sony Corporation Printed in Japan

3-795-898-51 (1)

SONY

MSX

Einführung in MSX-BASIC

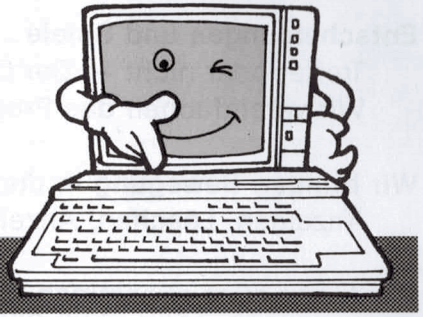


HiT BiT

Einführung in
MSX-BASIC

INHALTSVERZEICHNIS

EINFÜHRUNG



- Über dieses Buch
- Bello, der kluge Hund
- Der Sony Computer

ÜBER DIESES BUCH

Dieses Buch ist eine Einführung in die Programmiersprache BASIC.

Was ist eigentlich BASIC? Es ist eine einfache Sprache, in der man sich mit Computern „unterhalten" kann. Der Name BASIC kommt aus dem Englischen und bedeutet so viel wie **Grundlage**. BASIC ist also eine einfache, grundlegende Sprache, die ein Computer verstehen kann.

Aber warum wollen wir uns eigentlich mit einem Computer unterhalten? Ganz einfach!—Der Sony Computer beispielsweise läßt sich in vielen Bereichen des menschlichen Lebens nutzbringend einsetzen: Er kann Berechnungen ausführen, Informationen speichern, Probleme lösen, aber auch uns durch lustige Spiele die Zeit vertreiben. Mit diesem Buch werden wir schon nach einer oder zwei Stunden mit den wichtigsten Funktionen des Sony Computers vertraut sein. Je mehr wir ihn kennenlernen, um so mehr Freude wird er uns bereiten.

Zunächst werden wir lernen, wie der Computer unsere Anweisungen verstehen kann und wie wir uns mit ihm in der Sprache BASIC unterhalten können. Danach werden wir dann mit dieser speziellen Programmiersprache den Computer dazu bringen, einige interessante Kunststücke vorzuführen — erst ganz einfache und später dann auch etwas schwierigere. Und gegen Ende des Buches werden wir dann sogar in der Lage sein, den Sony Computer so zu programmieren, daß er zu einem Glücksspielapparat wird, wie er in ähnlicher Form auch in Spielhöllen zu finden ist.

Wenn wir das Buch erst einmal durchgearbeitet haben und BASIC keine „Fremdsprache" mehr für uns ist, werden wir nicht nur die in Computer-Zeitschriften abgedruckten Programme verstehen und anwenden, sondern auch mit Freunden, die das gleiche Hobby haben, fachsimpeln können. Wir erhalten mit diesem Buch genug Grundwissen, um später auch Bücher für Fortgeschrittene leicht verstehen zu können.

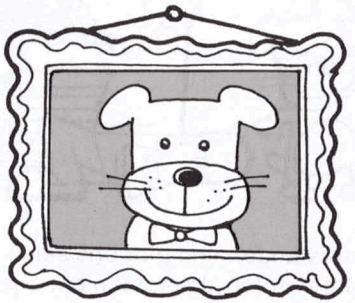
Um über oder mit Computern sprechen zu können, benötigt man viele Spezialwörter. Da wir uns als Anfänger diese Wörter sicher nicht gleich alle merken können — unser Gedächtnis arbeitet ja nicht so zuverlässig wie das des Computers — sind die wichtigsten Wörter mit den betreffenden Seitenzahlen noch einmal im **Register** hinten im Buch zusammengestellt.

Und noch etwas sollten wir uns immer vor Augen halten: Ein Computer hat niemals schlechte Laune. Er nimmt alles, wie es kommt, und es macht ihm überhaupt nichts aus, wenn wir beim Erlernen der Computersprache am Anfang viele Fehler machen. Wir brauchen also gar keine Angst zu haben — es kann nichts „passieren", auch wenn wir versehentlich einmal eine „falsche" Taste gedrückt haben. Genau wie bei jedem anderen Hobby ist es ganz natürlich, daß man am Anfang Fehler macht, aus den Fehlern dann lernt und so allmählich besser wird. Dabei werden wir dann bald zu schätzen wissen, daß der Computer uns immer wieder auf unsere Fehler aufmerksam macht, ohne ein einziges Mal ärgerlich zu werden. Er ist der geduldigste Lehrer, den man sich vorstellen kann.

Wir können uns den Sony Computer wie einen guten Freund vorstellen, der uns beim Programmieren hilft.

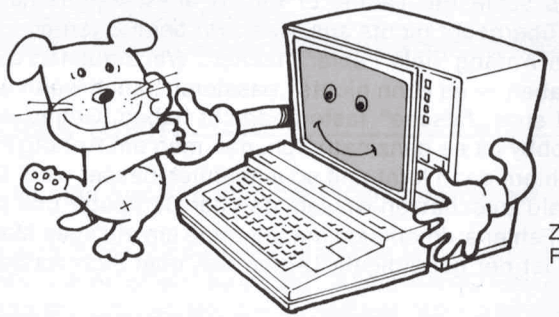
BELLO, DER KLUGE HUND

Beim Erlernen von BASIC wird uns noch ein Freund behilflich sein: Der Hund namens Bello. Er ist ein ganz normaler Hund und wie die meisten Hunde ist er freundlich, gutmütig und dazu noch ausgesprochen gelehrig. Er beherrscht eine Menge Kunststücke, die er uns jederzeit gerne vorführt.



Bello

Warum kommt Bello eigentlich in einem Buch über BASIC und Computer vor? Ganz einfach! Beide sind gute Freunde: Wie wir bald feststellen werden, macht es unserem Computer genau wie Bello sehr viel Spaß, **Befehle** auszuführen und Kunststücke vorzuführen. Indem wir die Reaktionsweise der beiden Freunde miteinander vergleichen, wird es uns leichter fallen zu verstehen, wie unser Computer die Befehle der Sprache BASIC versteht, wie er uns etwas mitteilt und wie er denkt.



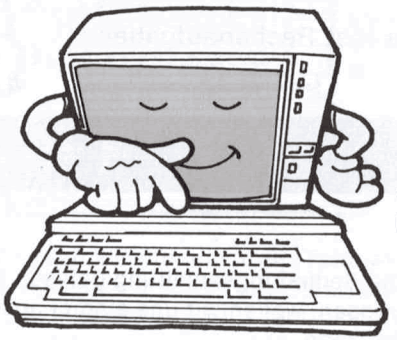
Zwei gute Freunde

DER SONY COMPUTER

Wir wissen schon, daß Bello eine ganze Menge Kunststücke beherrscht. „Sitz!", „Such!" und „Bring!" beherrscht er ebenso spielend wie „Leg dich!" — Und unser Sony Computer? Versteht er beispielsweise „Gib Pfötchen"?

Bello hat das sofort verstanden und gibt uns seine Pfote. Aber der Computer hat natürlich keine Pfoten. Ja er hat nicht einmal Ohren, mit denen er unseren Befehl hören könnte!

Aber wie können wir unserem Computer dann etwas mitteilen? Ganz einfach! Statt „Ohren" besitzt der Computer eine **Tastatur**, mit der wir ihm unsere Befehle geben können.



Wir lassen unsere Finger sprechen.

Bevor wir uns aber auf der nächsten Seite genauer mit der Programmiersprache BASIC befassen, sollten wir erst einmal die **Bedienungsanleitung** des Sony Computers durcharbeiten, um uns mit der Tastatur vertraut zu machen. Dabei ist es gar nicht notwendig, sich gleich die Funktion aller Tasten zu merken. Auf jeden Fall sollte uns aber klar sein, welche Funktion der **Cursor** hat und wie man ihn bewegt. Zur Übung versuchen wir einmal, einfach einige **Buchstaben, Zahlen** und **Grafiken** auf dem Bildschirm abzubilden.

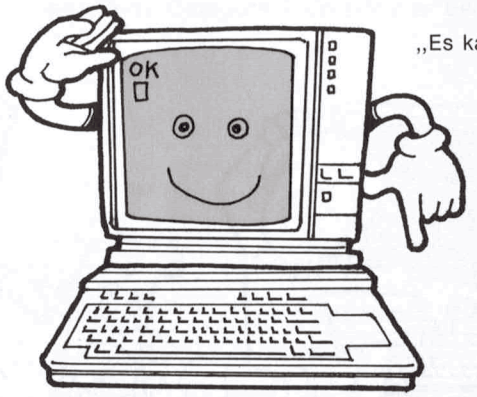
JETZT KANN ES LOSGEHEN

- Stoppen des Computers
- Befehlseingabe
- Farbige Zeichen
- Umschalttaste
- Unser Computer löst Rechenaufgaben

Nachdem wir nun die Bedienungsanleitung gelesen haben und über die Tastatur Bescheid wissen, wollen wir uns einmal anschauen, was unser Sony Computer alles kann.

BEFEHLE UND EINGABEN

Wie bereits erwähnt, können wir die Tastatur als das „Ohr" betrachten, mit dem der Computer unsere Mitteilungen versteht. Immer wenn der Computer bereit ist, uns „zuzuhören", gibt er uns dies freundlicherweise durch den Hinweis „Ok" auf dem Bildschirm bekannt.



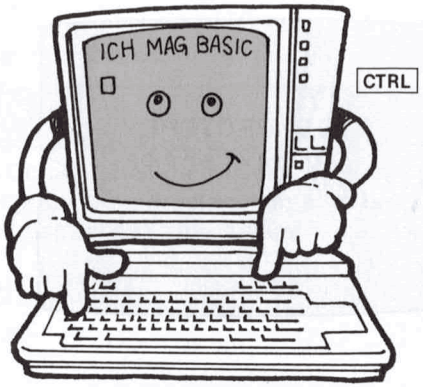
„Es kann losgehen!"

Alle mit der Tastatur eingegebenen Buchstaben und Zahlen erscheinen auf dem Bildschirm. Der Cursor bewegt sich dabei immer weiter von dem „Ok" weg.

Wir tippen ein paar Wörter ein und beobachten, wie sich der Cursor bewegt.

Alles klar? Dann drücken wir nun die CTRL-Taste mit einem Finger und gleichzeitig die STOP-Taste mit einem anderen. Wenn diese beiden Tasten **gleichzeitig** gedrückt werden, vergißt der Computer alles, was wir eingegeben haben und ist für unsere neuen Befehle bereit.

Unter Befehlen versteht man etwas anderes als unter **Eingabe**. Als Eingabe bezeichnet man die Buchstaben und Zahlen, die dem Computer mitgeteilt werden sollen—wie beispielsweise alle Buchstaben des Satzes „Ich mag BASIC". Mit dieser Eingabe alleine weiß der Computer jedoch noch nicht, was er eigentlich tun soll. Deshalb sind zusätzlich noch **Befehle** notwendig, die dem Computer entweder durch Drücken bestimmter Tasten oder durch Eintippen eines Befehlswortes mitgeteilt werden. Am besten läßt sich der Unterschied zwischen Eingabe und Befehl am Beispiel eines Taschenrechners verdeutlichen: Bei der Addition 2+2=4 handelt es sich bei „2", „+", „2" um Eingaben. Das Drücken der „="-Taste ist dagegen ein **Befehl**, der den Taschenrechner veranlaßt, die Addition 2+2 auszuführen und uns das Ergebnis mitzuteilen: 4. Genauso läuft es auch bei einem Computer ab. Und noch etwas sollten wir uns sehr gut merken: Wenn ein **Befehl** richtig eingegeben wurde, erscheint **immer** der Hinweis „Ok" direkt über dem Cursor und zeigt uns damit an, daß alles in Ordnung ist.

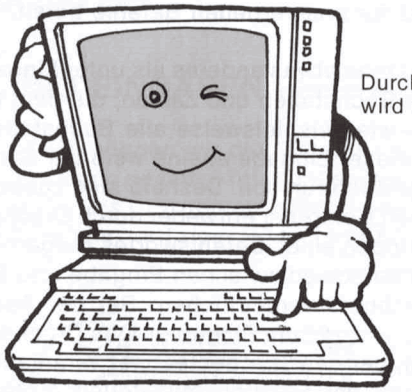


CTRL und STOP drücken.

Der Computer ist nun zur Eingabe von Befehlen bereit, die wahlweise mit Groß- **oder** Kleinbuchstaben erfolgen kann. Dann wollen wir einmal versuchen, ob unser Computer nun den Befehl „GIB PFÖTCHEN" verstehen und ausführen kann.

```
GIB PFÖTCHEN
```

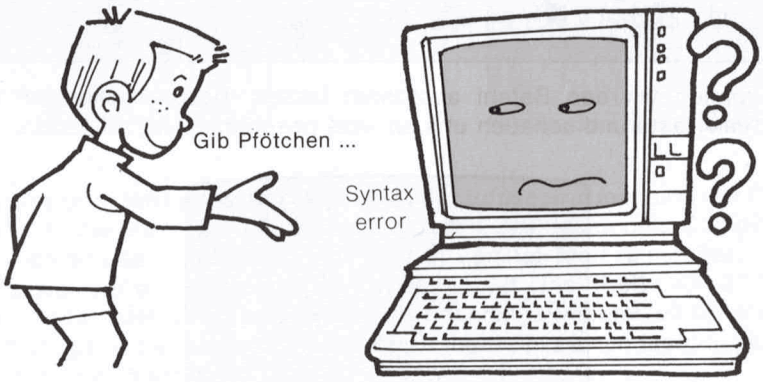
Der Computer reagiert nicht ...
Wir müssen dem Computer erst mitteilen, daß die Eingabe des Befehls abgeschlossen ist und er ihn ausführen soll. Zu diesem Zweck muß die Taste mit der Aufschrift RETURN gedrückt werden. Sie befindet sich ganz rechts und ist etwas größer als die anderen Tasten, da sie sehr oft verwendet wird. Diese RETURN-Taste muß jedes Mal gedrückt werden, **wenn der Computer den eingegebenen Befehl ausführen soll.**



Durch Drücken der RETURN-Taste wird der Befehl ausgeführt.

Wenn wir RETURN drücken, reagiert der Computer auf unseren Befehl. Allerdings gibt er natürlich nicht das Pfötchen, sondern meldet sich mit einem „Piepton" und teilt uns auf dem Bildschirm in etwa folgendes mit:

Hierbei handelt es sich um eine **Fehlermeldung**, d.h. der Computer hat zwar unseren Befehl gehört, ihn aber—wie nicht anders zu erwarten—gar nicht verstanden. Mit der Fehlermeldung „Syntax error" teilt uns der Computer mit, daß uns in der Programmiersprache BASIC ein Fehler unterlaufen ist.



Daß der gute alte Bello die Befehle „Sitz!", „Gib Pfötchen!" und vieles andere versteht, hängt natürlich damit zusammen, daß er entsprechend dressiert wurde. Unser Computer hat dagegen bei seiner „Dressur" ganz andere Befehle „gelernt"—und zwar die Befehle der Sprache BASIC. Wenn wir erst einmal diese Befehle gelernt haben, wird uns das Gespräch mit dem Computer keine Probleme mehr bereiten. Am besten wir fangen gleich damit an ...

EINIGE KUNSTSTÜCKE, DIE BELLO NICHT BEHERRSCHT

Am Anfang etwas ganz Einfaches:

```
WIDTH 10
```

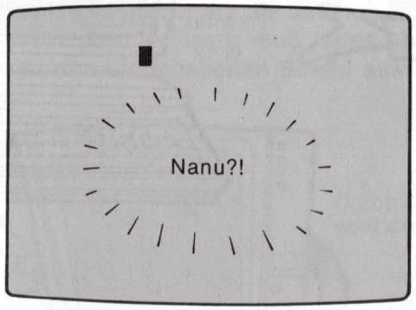
Das Eintippen in den Computer ist ganz leicht. Aber wir dürfen nicht vergessen, die Leertaste einmal zu drücken, um eine **Leerstelle** zwischen dem Wort WIDTH und der Zahl 10 zu erhalten.

W I D T H [ ] 1 0

Keine Sorge, wenn dabei einmal ein Tippfehler unterläuft. Wir brauchen lediglich den Cursor wie in der **Bedienungsanleitung** beschrieben zur betreffenden Stelle zu bewegen, und schon kann der Fehler korrigiert werden. Wenn auf dem Bildschirm folgendes erscheint, ...

```
WIDTH 10■
```

... können wir den Befehl ausführen lassen. Dazu drücken wir die RETURN-Taste und schauen uns an, was passiert.



Alle Ziffern sind verschwunden und der Cursor befindet sich nahe der Bildschirmmitte. Und was hat der Befehl WIDTH 10 nun für eine Bedeutung? Um dies zu verstehen, geben wir am besten einmal das ganze Alphabet ein, ohne Leerstellen und ohne Drücken der RETURN-Taste.

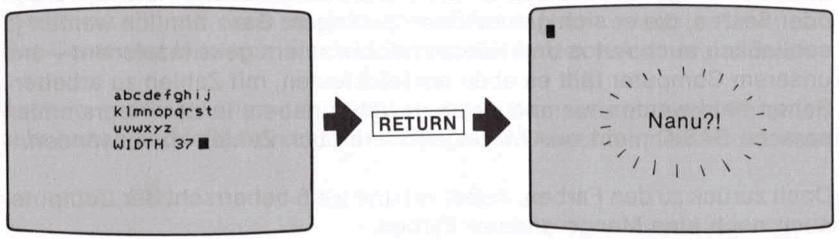
```
abcdefghij
klmnopqrst
uvwxyz■
```

In einer Zeile befinden sich also zehn Buchstaben. Dies hat offensichtlich der Befehl WIDTH 10 bewirkt. Wir haben damit schon den ersten BASIC-Befehl gelernt: Wenn wir WIDTH 10 eingeben, macht der Computer jede Zeile genau **zehn Zeichen** breit.

Dann schauen wir uns gleich den nächsten Befehl an — aber zuvor drücken wir gleichzeitig die CTRL- und STOP-Taste, um die letzten Eingaben zu „löschen" und den Computer auf den **nächsten** Befehl vorzubereiten.

```
WIDTH 37
```


Nun drücken wir die RETURN-Taste.



Was ist denn jetzt passiert? Wieder ist der Bildschirm leer, und der Cursor ist in die linke obere Ecke zurückgekehrt. Wir haben dem Computer nämlich den Befehl gegeben, jede Zeile 37 Zeichen breit zu machen, und alles, was wir nun eintippen, wird in Zeilen mit einer Breite von 37 Zeichen abgebildet. (Jedesmal wenn wir unseren Computer einschalten, wird übrigens die Zeilenlänge automatisch auf 37 Zeichen festgelegt, es sei denn, wir ändern die Länge.)

Dann probieren wir gleich auch noch einmal einen anderen Befehl aus, den Bello natürlich auch nicht verstehen würde. Zunächst drücken wir wieder CTRL und STOP und dann geben wir folgendes ein:

```
COLOR 8
```

(Aber nicht vergessen, die RETURN-Taste zu drücken!) Jetzt wird plötzlich alles, was wir eintippen, rot auf dem Bildschirm abgebildet.

Wie wäre es mit einer anderen Farbe? Zunächst bringen wir den Cursor wieder durch Drücken von CTRL und STOP zur linken oberen Bildschirmecke zurück. Dann geben wir COLOR 15 ein, drücken die RETURN-Taste —und schon wird wieder alles weiß auf dem Bildschirm abgebildet.

15 bedeutet also weiße Farben und 8 rote Farben. Aber warum kann man nicht einfach die englischen Wörter für rote Farbe, also COLOR RED oder weiße Farbe, also COLOR WHITE eingeben? Für Menschen und auch für unseren Hund Bello wäre dies natürlich einfacher zu verstehen; unser Computer dagegen tut sich leichter mit Zahlen. Wie wir später noch sehen, werden in der Programmiersprache BASIC des öfteren Wörter einfach durch Zahlen ersetzt.

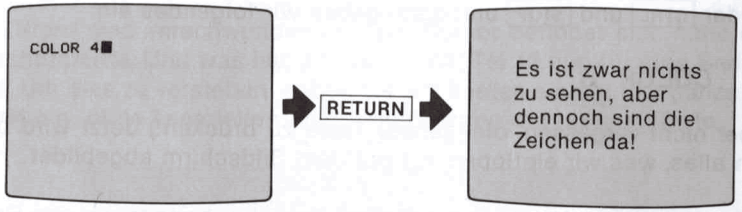
Wenn wir erst einmal mit der Arbeitsweise unseres Computers etwas weiter vertraut sind, wird es uns kein Problem mehr bereiten, den Computer durch Eingabe von Zahlen nach bestimmten Namen, Plätzen, Farben oder Sätzen, die er sich gemerkt hat, zu fragen. Ganz ähnlich werden ja schließlich auch Autos und Häuser mit Nummern gekennzeichent—und unserem Computer fällt es eben am leichtesten, mit Zahlen zu arbeiten. Schon bald werden wir uns daran gewöhnt haben, in der Programmiersprache BASIC nicht nur Wörter, sondern auch Zahlen zu verwenden.

Doch zurück zu den Farben. Außer rot und weiß beherrscht der Computer auch noch eine Menge anderer Farben.

Code	Farbe	Code	Farbe	Code	Farbe	Code	Farbe
0	Transparent	4	Dunkelblau	8	Rot	12	Dunkelgrün
1	Schwarz	5	Hellblau	9	Hellrot	13	Magentarot
2	Mittelgrün	6	Dunkelrot	10	Dunkelgelb	14	Grau
3	Hellgrün	7	Himmelblau	11	Hellgelb	15	Weiß

Insgesamt haben wir 16 Farben zur Verfügung, die wir nach Belieben einsetzen können. Z.B. können wir folgendes versuchen:

```
COLOR 4
```



Dunkelblaue Zeichen auf dunkelblauem Hintergrund.
Da ist natürlich nichts zu erkennen...

Wir können also nun die Farbe unserer Zeichen nach Lust und Laune wählen. Am besten probieren wir mal die ganze Farbentabelle durch.

Nun wird es etwas trickreicher ...

```
COLOR 1000
```


(Die RETURN-Taste nicht vergessen!) Die Zahl 1000 kommt gar nicht in unserer Farbentabelle vor, und der Computer weiß das natürlich. Sofort protestiert er mit einem Piepton und teilt uns auf dem Bildschirm „Illegal function call" mit, was so viel bedeutet wie „Nicht erlaubte Funktion". Der Computer läßt sich also von uns nicht an der Nase herumführen.

Auch in anderen Fällen nimmt es der Sony Computer mit den Befehlen sehr genau und läßt da nicht mit sich spaßen. Angenommen, wir wollten

```
COLOR 6
```

eintippen, haben aber, ohne es zu merken,

```
CILOR 6
```

eingetippt. Wird nun die RETURN-Taste gedrückt, so meldet sich der Computer wieder mit einem Piepton und auf dem Bildschirm erscheint die schon bekannte Fehlermeldung „Syntax error".

Natürlich macht jeder einmal einen Fehler—selbst die erfahrendsten Programmierer bei Sony! Deshalb besitzt die Tastatur eine Rücksetztaste BS, eine Löschtaste DEL, eine Einfügtaste INS und Cursor-Steuertasten, mit denen wir Tippfehler ganz einfach korrigieren können.

Wie viele BASIC-Wörter gibt es?
Zwei BASIC-Befehle WIDTH und COLOR sind uns nun schon bekannt. Wir wissen auch, wie genau es der Sony Computer mit den Befehlen nimmt: Er führt sie aus, wenn sie richtig eingegeben werden und warnt mit einem Piepton, wenn ein Fehler unterläuft oder er den Befehl nicht kennt. Doch wie viele BASIC-Befehle gibt es denn insgesamt? Etwa 100! Das reicht aus, um den Computer beispielsweise musizieren, Bilder malen und Rechenaufgaben lösen zu lassen. Im MSX-BASIC-Benutzerhandbuch ist dies alles genau erklärt. Doch keine Angst—wir brauchen gar nicht alle 100 Befehle zu kennen, es reichen schon ein paar, um mit dem Programmieren beginnen zu können.

Beim Programmieren meldet sich auch der Sony Computer manchmal „zu Wort". Er kennt 35 verschiedene **Fehlermeldungen**, die auf dem Bildschirm erscheinen und durch die er uns sagt, was er nicht verstanden hat. Zwei davon „Syntax error" und „Illegal function call" sind uns ja schon bekannt. Diese Fehlermeldungen sind gerade für Anfänger eine große Hilfe.

ABBILDEN DER ANTWORT

Bello mag einige Kunststücke wie „Gib Pfötchen" besonders gern. Die Lieblingsbeschäftigung unseres Sony Computers ist das Rechnen, und da ist er wirklich unschlagbar.

Wir geben zum Beispiel ein:

```
PRINT 3+5
```

Haben Sie das Pluszeichen (+) gefunden? Es befindet sich über dem „=". Um es auf dem Bildschirm abzubilden, müssen wir die SHIFT-Taste und **gleichzeitig** die „="-Taste drücken.

Zur Eingabe des Befehls in den Computer drücken wir dann die RETURN-Taste.

```
PRINT 3+5
 8
Ok
■
```

Da haben wir also das Ergebnis! Der Computer hat die Zahlen für uns zusammengezählt und das Ergebnis auf dem Bildschirm abgebildet. Dann versuchen wir es gleich mit einer anderen Rechenaufgabe:

```
PRINT 100-10
```

Für das Minuszeichen braucht die SHIFT-Taste nicht gedrückt zu werden.

Nun drücken wir wieder die RETURN-Taste zur Eingabe des Befehls.

```
PRINT 3+5
 8
Ok
PRINT 100-10
 90
Ok
■
```

Das Ergebnis ist natürlich 90. Genau wie diese einfachen Rechenbeispiele kann der Computer auch viel schwierigere Rechenaufgaben im Handumdrehen bewältigen.

Wie steht es nun mit dem Multiplizieren? Auch das ist kein Problem. Wir müssen lediglich wissen, daß der Computer hierzu ein **spezielles Zeichen** verwendet: Statt **7×9** geben wir **7 * 9** ein.

Wir tippen also PRINT 7 * 9 ein und drücken dann die RETURN-Taste wie gewohnt.

```
PRINT 7*9
 63
Ok
■
```

Zum Dividieren verwenden wir die /-Taste. Statt PRINT 120:40 müssen wir also eintippen.

```
PRINT 120/40
```

Und auch hier hat der Computer natürlich umgehend die Lösung parat, wenn wir die RETURN-Taste drücken.

```
PRINT 120/40
 3
Ok
■
```

Auch große Zahlenbeträge (z.B. 7654321×18) und etwas kompliziertere Rechenaufgaben mit Klammerausdrücken wie z.B. (2+9)×(6−8) bereiten jetzt kein Problem mehr. Am besten probieren wir einmal einige Rechenaufgaben durch, um uns mit dem ganzen vertraut zu machen. Dabei kann uns die Antwort des Computers zwar manchmal etwas unverständlich vorkommen, dies sollte uns aber nicht weiter beunruhigen. Gehen Sie langsam Schritt für Schritt vor und überlegen Sie sich immer gut, was Sie tun; dann werden Sie in Kürze mit dem Computer vertraut sein und auch schwierigere Rechenaufgaben lösen können.

Wenn es ums Rechnen geht,
ist der Computer immer Sieger.

ZAHLEN, BUCHSTABEN UND VARIABLEN

- Ausdruck
- Buchstaben werden zu Variablen
- Verwendung der Variablen in mathematischen Gleichungen
- Namensgebung der Variablen

Mit dem PRINT-Befehl konnte unser Computer Rechenaufgaben knacken, ganz ähnlich wie ein herkömmlicher Taschenrechner. Bevor wir uns nun anschauen, was unser Computer sonst noch alles ausrechnen kann, wollen wir erst einmal den PRINT-Befehl etwas genauer kennenlernen. Wir geben dazu einmal folgendes ein:

```
PRINT "INGE UND PETER"
```

Die **Anführungszeichen (")** befinden sich über dem Apostroph ('), so daß auch hier wieder die SHIFT-Taste gleichzeitig gedrückt werden muß. Nach der Eingabe drücken wir dann wieder die RETURN-Taste.

```
PRINT "INGE UND PETER"
INGE UND PETER
Ok
■
```

Der Computer hat natürlich die Anführungszeichen „gelesen", sie sind aber dann bei der Ausführung des PRINT-Befehls nicht auf dem Bildschirm erschienen. Die Anführungszeichen teilen dem Computer lediglich mit, daß er alles genauso abbilden soll, wie es zwischen die Anführungszeichen eingetippt wurde. Hier einige Beispiele dazu:

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

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


Auch hier dürfen wir natürlich nicht vergessen, jedesmal die RETURN-Taste zu drücken. Aber das wissen wir ja schon. Ab jetzt wird deshalb nicht mehr speziell darauf hingewiesen.

ZAHLEN ODER BUCHSTABEN—FÜR DEN COMPUTER IST DAS GLEICH

Nun wollen wir uns etwas anderem zuwenden.

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

Auf den ersten Blick sieht das genauso aus wie die Rechenaufgabe, die wir einige Seiten vorher gelöst haben. Wir erinnern uns: wenn wir **PRINT 3+5** eingeben, liefert uns der Computer die Lösung, also 8. — Allerdings nur wenn keine Anführungszeichen vorhanden sind ...

Mit Anführungszeichen sieht es dagegen ganz anders aus:

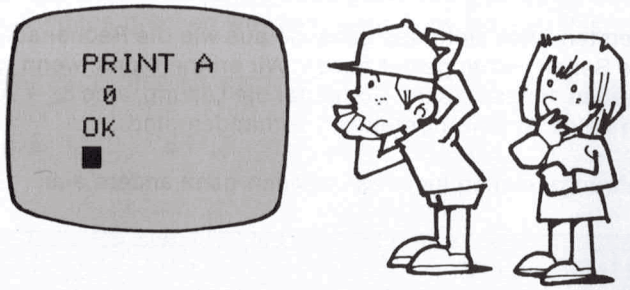
```
PRINT "3+5"
3+5
Ok
■
```

Wenn der **PRINT**-Befehl wie in diesem Fall zusammen **mit Anführungszeichen** verwendet wird, so bildet der Computer einfach den zwischen den Anführungszeichen stehenden Inhalt unverändert ab. **"3+5"** wird also vom Computer einfach als drei Zeichen behandelt—und nicht als Rechenaufgabe, die gelöst werden soll.

Der Computer behandelt alles gleich, was zwischen zwei Anführungszeichen steht.

EIN BUCHSTABE WIRD ZU EINER VARIABLEN

Dann wollen wir gleich noch etwas Interessantes über den PRINT-Befehl lernen. Was passiert, wenn wir einen Buchstaben ohne Anführungszeichen nach PRINT eintippen?



Was ist denn jetzt passiert? Kein warnender Piepton und keine „Syntax error"-Meldung! Statt dessen erscheint einfach 0 und dann Ok. Der Computer hat also unseren Befehl ausgeführt und wartet auf den nächsten Befehl. Aber welchen Befehl haben wir ihm eigentlich gegeben?

Da wir keine Fehlermeldung erhielten, muß der Computer den Befehl PRINT A kennen. Was wird nun damit bewirkt? Um dies herauszufinden, probieren wir einmal folgendes:

```
LET A=3
```


Wird die RETURN-Taste gedrückt, kommt zwar keine Fehlermeldung, aber es passiert auch nicht sehr viel, außer daß ein beruhigendes Ok erscheint. Unser Befehl muß also im Computer irgendetwas bewirkt haben.

Was spielt sich in diesem Fall wohl im Computer ab?

Nun wollen wir einmal die folgenden Befehle nacheinander eingeben:

```
LET A=3
Ok
PRINT A
 3
Ok
■
```

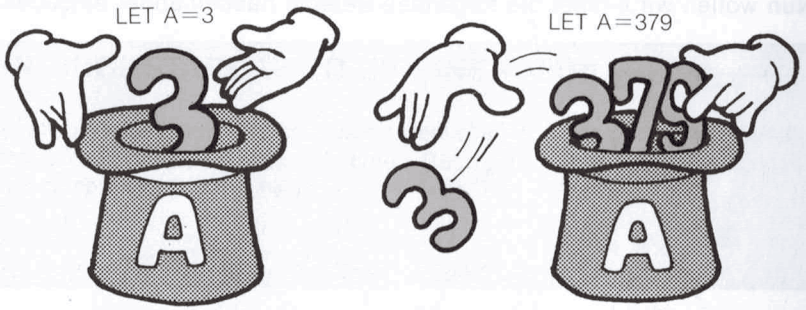
Vielleicht ist einigen schon klar, was hier passiert. Um ganz sicher zu gehen, probieren wir es noch einmal:

```
LET A=379
Ok
PRINT A
 379
Ok
■
```

Im ersten Beispiel war **A** gleich **3** und jetzt **379**. Wir können also A eine beliebige Zahl zuordnen, wenn wir erst **LET A=** und dann die gewünschte Zahl eingeben. A ist variabel und wird deshalb als **Variable** bezeichnet.

Wenn **A** ohne Anführungszeichen nach PRINT eingegeben wird, so wird es vom Computer gar nicht mehr als richtiger Buchstabe behandelt. Es ist dann wie in Behälter, in den wir eine beliebige Zahl hineintun können.

Die Funktion der Variablen können wir uns ganz gut mit dem Hut eines Zauberkünstlers verdeutlichen. Auch der Zauberer kann etwas in seinen Hut hineintun und dann später plötzlich wieder hervorzaubern.



Wenn wir also LET A=3 eintippen, haben wir die Zahl 3 im Hut, und wir können sie wieder hervorzaubern, indem wir PRINT A eintippen. Und mit einem einzigen Befehl (LET A=379) können wir die Zahl im Hut ändern und auch diese dann zu einem beliebigen Zeitpunkt wieder hervorholen.

Mit dem LET-Befehl können wir also unserer Variablen einen beliebigen Wert zuordnen. Variablen werden in BASIC für verschiedene Zwecke verwendet und kommen deshalb in unserem Buch noch öfter vor. Ja, es geht sogar noch einfacher: Wir können das Wort LET weglassen und nur folgendes eingeben:

```
A=3
```

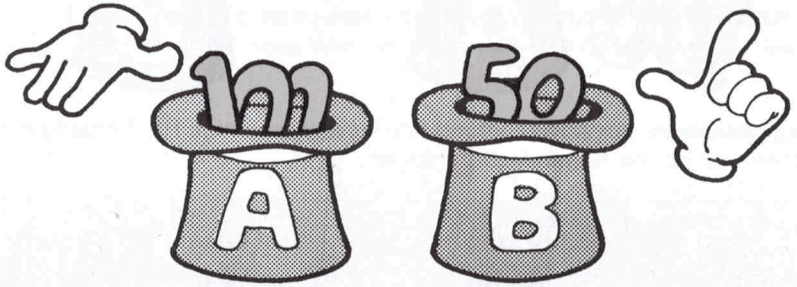
Dies hat die gleiche Bedeutung wie LET A=3.

Um sicherzustellen, daß wir auch nichts vergessen haben, wollen wir hier noch einmal alles kurz wiederholen:

```
A=100          Teile A den Wert 100 zu!
Ok
PRINT A        Bilde A ab!
 100           Wert der Variablen
Ok
PRINT "A"      Bilde den Buchstaben „A" ab (nicht die Variable A)!
A              Abgebildeter Buchstabe
Ok
B=50           Teile B den Wert 50 zu!
Ok
PRINT B        Bilde B ab!
 50            Der B zugeteilte Wert
Ok
■
```

Wir erinnern uns, daß der Computer die Zahl 0 abbildete, als wir am Anfang dieses Abschnittes den Befehl PRINT A eingaben. Das heißt also, daß den Variablen stets der Wert 0 zugeteilt wird, wenn wir keine andere Zuteilung vornehmen.

Bisher haben wir als Variablen A und B verwendet, aber natürlich eignen sich auch alle anderen Buchstaben.



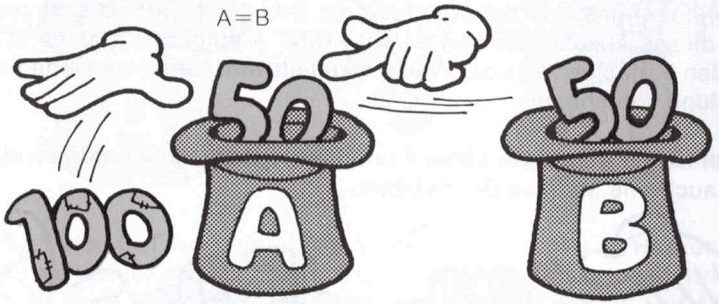
Aber was können wir mit diesen Variablen überhaupt anfangen?

Wir haben bereits A=100 und B=50 in den Computer eingegeben. Nun wollen wir noch folgendes eingeben: A=B, PRINT A und PRINT B. Auch wenn es vielen vielleicht schon klar ist, was passiert, wollen wir den Computer einmal die Befehle ausführen lassen.

```
A=B
Ok
PRINT A
 50
Ok
PRINT B
 50
Ok
■
```

Wir betrachten uns einmal die erste Zeile des Programms. Durch den Befehl A=B wurde A der Wert von B zugeteilt. Früher haben wir schon einmal den Wert von A von 3 zu 379 geändert, wobei der alte Wert einfach gelöscht wurde.

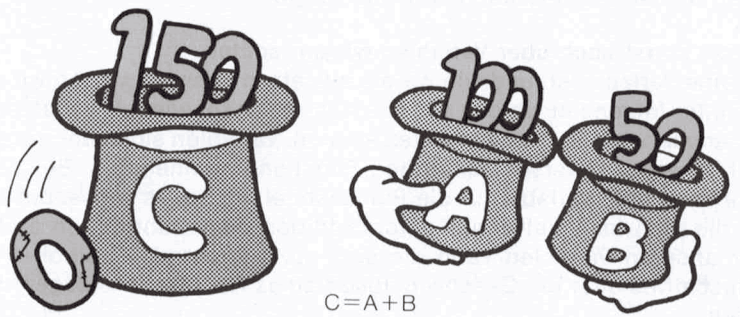
Genau das gleiche ist hier wieder passiert: Der Variablen A, die ursprünglich den Wert 100 besaß, wird der Wert von B, also 50, zugeteilt.



Nun wollen wir zur Abwechslung wieder einmal einige Rechenaufgaben lösen und geben folgende Befehle ein:

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

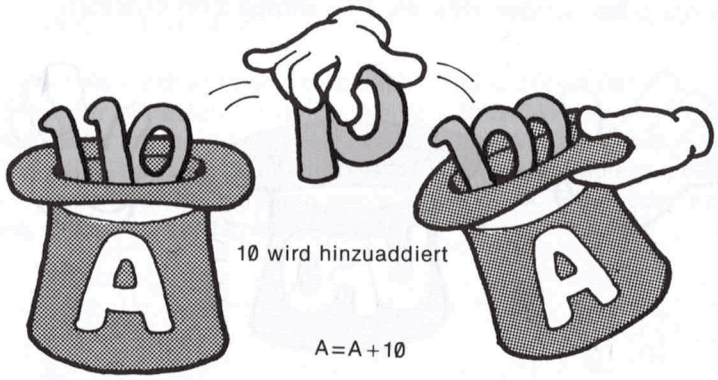

Hier kommen jetzt drei „Behälter" — A, B und C — vor, und der Computer rechnet aus, welcher Wert in den Behälter C hineingehört.



Noch ein interessantes Programm:

```
A=A+10
Ok
PRINT A
 110
Ok
■
```

Auf den ersten Blick scheint dies etwas verwirrend. Doch mit etwas Überlegung wird alles klar: A war zuvor der Wert 100 zugeordnet worden. Durch das Programm wird zu dem alten Wert von A die Zahl 10 hinzuaddiert, so daß sich der neue Wert von A zu 110 berechnet.



Den gleichen Befehl könnten wir nun noch öfter ausführen lassen. Dabei müssen wir uns immer vor Augen halten, daß der Computer den einer Variablen einmal zugeordneten Wert **nie vergißt**.

Was wir sonst noch über Variablen wissen sollten

Was eine Variable ist, und wie wir sie einsetzen können, wissen wir nun schon. Im folgenden werden wir uns nun ansehen, wie wir Variablen in Programmen und Spielen einsetzen können. Variablen sind sehr nützlich und können ganz verschiedenartige Funktionen einnehmen. Beispielsweise kann eine Variable für die Punktzahl eines Spiels verwendet werden, die sich im Spielverlauf durch Addition oder Subtraktion ändert. Einer anderen Variablen kann die sich stets ändernde Position eines Raumschiffes oder die Geschwindigkeit eines Rennwagens zugeordnet werden.

Bisher haben wir für die Variablen lediglich die drei Buchstaben — A, B und C—verwendet. Aber natürlich können wir auch **beliebige** andere Buchstaben oder Buchstabengruppen verwenden.

```
UFO=5
Ok
PRINT UFO
 5
Ok
■
```

Wie wir sehen, kann eine Gruppe aus drei Buchstaben, die ohne Leerstellen eingegeben werden müssen, eine **einzige** Zahl bedeuten.

Dabei müssen wir uns immer daran erinnern, daß unterschiedliche Buchstaben oder Buchstabengruppen, also z.B. A, B oder AB stets unterschiedliche Bedeutung haben.

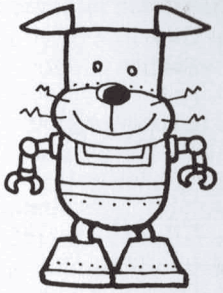

```
A=3
Ok
B=5
Ok
PRINT A
 3
Ok
PRINT B
 5
Ok
PRINT AB
 0
Ok
■
```

A wurde hier der Wert 3 und B der Wert 5 zugeordnet — wogegen der Variablen AB kein Wert zugeordnet wurde, so daß der Computer für AB den Wert 0 (**nicht** 35!) abbildet. Einer Variablen wird stets der Wert 0 zugeordnet, wenn wir nichts anderes festlegen.

Variablen können sich aus 2, 3, 10 — oder sogar 100 — **Zeichen** zusammensetzen. Für diese Zeichen können wahlweise **Zahlen** oder **Buchstaben** verwendet werden. Allerdings **liest der Computer nur die ersten beiden Zeichen, und das erste Zeichen muß auf jeden Fall ein Buchstabe sein**. Für den Computer ist POA, POB, PO1 und PO2 genau das gleiche. Wir müssen deshalb etwas vorsichtig sein, wenn wir uns einen Namen für unsere Variable ausdenken.

Wörter oder Teile der Programmiersprache BASIC dürfen natürlich **nicht** als Namen für Variablen verwendet werden. BASIC-Wörter wertet unser Sony Computer immer nur als **Befehl** und nicht als Variable. Das bedeutet beispielsweise, daß LET, GOTO und PRINT nicht als Variablennamen verwendet werden dürfen, genauso wenig wie GOTOA, BLET oder andere Befehle, die wir später lernen werden.

WIR SCHREIBEN UNSER ERSTES PROGRAMM



- Planen eines Programms
- Schreiben eines einfachen Programms
- Speichern des Programms im Computer
- Korrigieren eines Programms
- Programmlauf
- Auflisten des gesamten Programms auf dem Bildschirm
- Löschen des Programms aus dem Speicher des Computers

Die grundlegende Wirkungsweise unseres Computers ist uns nun schon bekannt: Wir können Zeichen, Zahlen und Symbole eingeben, die der Computer versteht, und können ihm einfache Anweisungen (Befehle) geben. Mit anderen Worten, wir können den Computer etwas auf dem Bildschirm so **auszuführen** lassen, wie wir es haben möchten.

Doch bisher hätten wir im Grunde alles auch von einem ganz einfachen Taschenrechner machen lassen können. Ein Taschenrechner kann schließlich auch u.a. addieren und subtrahieren und uns alle Schritte sowie das Ergebnis auf seinem Display anzeigen.

Doch ein **Computer** kann noch viel mehr—und das macht ihn für uns, genau wie für Wissenschaftler und viele andere Leute, so interessant:

- Abspeichern einer Serie von **Spezialanweisungen, die wir ihm gegeben haben**, im **Memory**.
- Mit den Anweisungen können bestimmte Aufgaben, Spiele usw. zu einer beliebigen Zeit ausgeführt werden.
- Ein bestimmtes Ergebnis auswerten und daraus bestimmte nächste Schritte einleiten.

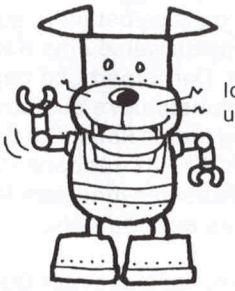
Mit anderen Worten, unser Computer kann ein ganzes **Programm** von Funktionen automatisch abarbeiten und zwar viel schneller, als dies ein Taschenrechner könnte.

Unter **Programmieren** versteht man die Eingabe von Anweisungen in den Computer, um ihm zu sagen, was er in welcher Reihenfolge ausführen soll. Nur dadurch ist es möglich, daß wir unseren Computer für Videospiele, Zeichnen von Bildern und sogar zum Erzeugen von elektronischer Musik einsetzen können. Damit wollen wir uns nun im folgenden Kapitel beschäftigen: Wir erstellen unser erstes, einfaches Programm.

Das ist gar nicht so schwierig. Wie wir unserem Computer Befehle geben, wissen wir ja schon, und ein Programm ist im Grunde nur eine Aneinanderreihung von Befehlen. Wichtig dabei ist die **Reihenfolge** der Befehle, und diese wiederum wird durch eine **Numerierung** festgelegt.

Bevor wir uns mit dem Programmieren etwas genauer beschäftigen, wollen wir einen anderen, sehr klugen Hund vorstellen, der uns helfen wird, die Vorgänge leichter zu verstehen. Bello hat also nun Pause, und der **Superhund** wird uns zeigen, was er alles kann.

Der Superhund ist nicht nur gutmütig, sondern auch klug und gelehrig: Er ist ein Roboter! (Warum auch nicht! Alles wird ja heutzutage automatisiert.)
tisiert.)



Ich bin der Superhund
und kann ein ganzes Programm ausführen.

Bello beherrscht eine ganze Menge Kunststücke, wie wir ja schon wissen. Wenn wir einen Ball werfen, rennt er los, sucht ihn und bringt ihn uns zurück. Und wenn wir zu ihm „Gib Pfötchen" sagen, hebt er seine Pfote und läßt sie uns schütteln. Wir mögen ihn, weil er all dies kann. Aber eigentlich kann er ja bei jedem Kunststück immer nur eine Abfolge von drei oder vier Befehlen ausführen. Eine längere Abfolge kann er sich nicht merken. Wenn er ein Kunststück beendet hat, müssen wir ihn wieder mit einem neuen „programmieren".
Und was kann der Superhund?

PLANEN EINES PROGRAMMS

Der Superhund kann noch viel mehr. Er ist ein Mikroprozessor, der sich auch längere Vorgänge merken und sogar Entscheidungen fällen kann. Aufgrund dieser Fähigkeit kann er sehr viel mehr Schritte selbständig ausführen, als unser Hund Bello. Der Superhund ist Bello bei weitem überlegen. So können wir unseren Superhund beispielsweise folgendes bitten:

1. Such den Ball und bringe ihn zurück!
2. Ist der Ball weiß, drehe dich dreimal herum und belle!
3. Ist der Ball schwarz, lege dich zehn Sekunden hin!

Den ersten Schritt würde auch Bello ohne weiteres ausführen können. Doch beim zweiten und dritten Schritt müßte er passen. Die Farbe eines Balles erkennen und dann danach entscheiden, welches Kunststück ausgeführt werden muß, das würde Bello überfordern.

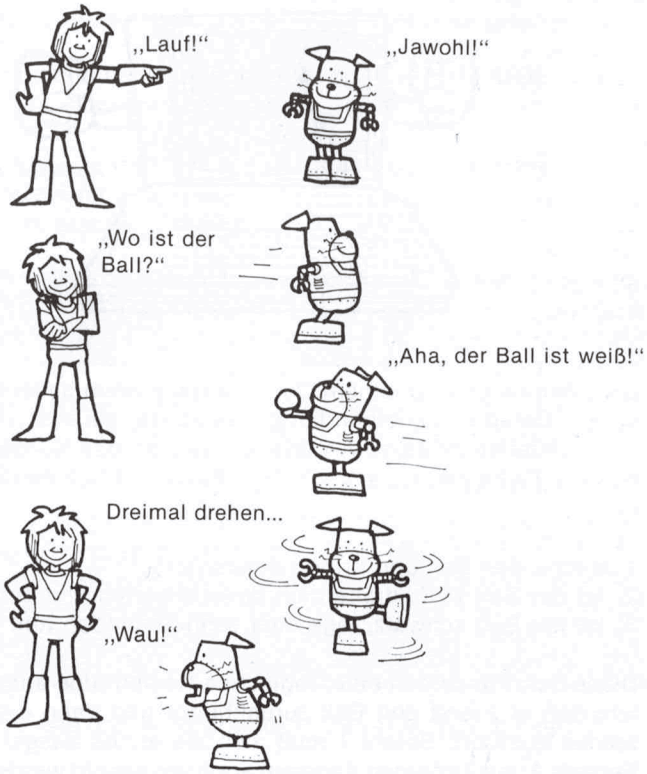
Außerdem müßte Bello bei diesem Kunststück zählen können: Er müßte genau wissen, wievielmal er sich drehen bzw. wie lange er sich hinlegen soll. Unser Superhund kann selbst etwas beurteilen, Entscheidungen fällen und sogar beim Ausführen seiner Kunststücke etwas ausrechnen.

Und noch ein wichtiger Unterschied besteht zwischen Bello und dem Superhund. Bello ist ein ganz normales Tier, dessen Gehirn ähnlich wie das des Menschen ist. Er kann also etwas ganz selbständig ausführen, ohne daß er Befehle erhält. Wenn Bello beispielsweise eine Katze sieht, bellt er; wenn er Feuer sieht, läuft er davon. Der Superhund dagegen ist eine Maschine, die zwar sehr komplizierte Kunststücke ausführen, aber nicht „denken" und auch nicht aus Erfahrung lernen kann. Der Superhund handelt **nur** so, wie er **programmiert** wurde. Er tritt nur dann in Aktion, wenn wir ihm befehlen, mit seinen Motoren, Sensoren, und dem Mikroprozessor in einer bestimmten Reihenfolge etwas auszuführen.

Wenn wir also ein Programm schreiben, nach dem der Superhund seine Kunststücke ausführen soll, müssen wir uns vorher erst alles—d.h. **jeden** kleinen Schritt und auch die Reihenfolge der Schritte—ganz genau durchdenken. Beispielsweise können wir folgende Überlegung anstellen:

1. Nimm den Befehl „Ball suchen und zurückbringen" auf!
2. Verfolge mit deinem Bildsensor den Ball und orte ihn!
3. Bewege dich mit deinen mechanischen Beinen zum Ball!
4. Stelle mit deinem Bildsensor fest, ob der Ball „schwarz" oder „weiß" ist!
5. Bewege deine Körperglieder so, daß du den Ball aufnimmst und zurückbringst!
6. Entscheide, ob du dich „herumdrehen" oder „hinlegen" mußt!
7. Bewege deine Körperglieder so, daß die geforderte Aktion ausgeführt wird!
8. Berechne die Zahl der Drehungen bzw. die Zeit!
9. Belle bzw. belle nicht!
10. Stoppe die Bewegungsabläufe, wenn das Kunststück fertig ist!
11. Bereite dich auf das nächste Programm vor!

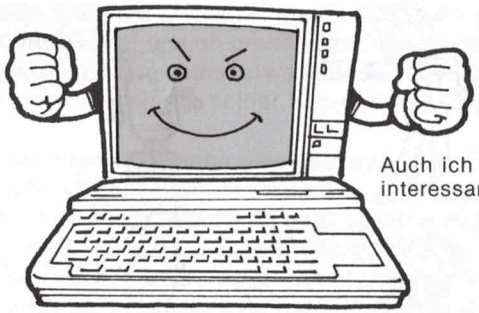
Wenn wir bei sämlichen Schritten genau geprüft haben, ob nichts vergessen wurde, ob kein Fehler gemacht wurde und ob die Reihenfolge stimmt, können wir alles beruhigt in den Mikroprozessor des Superhundes eingeben. Er wird dann alles geduldig wie gewünscht ausführen, ganz egal, ob wir das Programm einmal, zehnmal oder zehntausendmal „laufen lassen".



Zwei Hauptregeln gilt es also beim Schreiben eines Programms immer zu beachten: **Alles** genau überlegen, was der Computer ausführen soll und alle Schritte in der **richtigen Reihenfolge** anordnen. Wenn wir immer nur ganz normal und logisch denken, kann im Grunde gar nichts schiefgehen. Denn der Computer arbeitet wie jede andere Maschine nach den Regeln der Logik.

Jetzt wollen wir aber endlich unser erstes Programm mit unserem Sony Computer schreiben. Es ist wirklich nicht schwer, wie wir gleich sehen werden.

EIN PROGRAMM SCHREIBEN—KEIN PROBLEM —



Auch ich kann einige ganz interessante Kunststücke!

Die „Reihenfolge" in einem Computerprogramm bedeutet lediglich, daß wir die Befehle nacheinander, genau so wie sie ausgeführt werden sollen, aufstellen müssen. Wir erinnern uns an das Kunststück des Superhundes: Es bestand aus drei Teilen, die wir mit Nummern gekennzeichnet hatten.

1. Suche den Ball und bringe ihn zurück!
2. Ist der Ball weiß, drehe dich dreimal herum und belle!
3. Ist der Ball schwarz, lege dich zehn Sekunden hin!

Diese Schritte sind in einer logischen Reihenfolge angeordnet: Wir wollen, daß er zuerst den Ball zurückbringt und dann die anderen Kunststücke ausführt; Befehl 1 muß also als erstes ausgeführt werden (die Befehle 2 und 3 könnten dagegen auch vertauscht werden, ohne daß sich am Kunststück selbst etwas ändert.)

Die Eingabe der Befehle in den Computer erfolgt in gleicher Weise: Wir schreiben sie in der Reihenfolge auf den Bildschirm, in der sie ausgeführt werden sollen, und geben jedem Befehl eine Nummer. Die erste „Aufgabe" unseres Computers, also das erste Programm, wird lauten: Schreibe „INGE UND PETER" auf den Bildschirm.

```
10 PRINT "INGE"
20 PRINT "UND"
30 PRINT "PETER"
```

Den PRINT-Befehl, den wir nun ja schon kennen, verwenden wir hier dreimal—einmal für jedes der drei Wörter, Die Reihenfolge ist natürlich genau wie die Reihenfolge der Wörter. Der einzige Unterschied zu der früheren Verwendung des PRINT-Befehls besteht darin, daß nun jeder Befehl mit einer Nummer beginnt: 10, 20 und 30. Diese Nummern sind sehr wichtig; sie sagen dem Computer **welcher Befehl des Programms als nächster** ausgeführt werden soll, d.h. sie geben die Reihenfolge an.

Dann geben wir nun mal den ersten Befehl ein: 10 PRINT "INGE"; natürlich dürfen wir nicht vergessen, RETURN zu drücken!

1 0 [ ] P R I N T [ ] " I N G E " RETURN

Auf dem Bildschirm erscheint dann ...

```
10 PRINT "INGE"
■
```

Gut! Der erste Befehl ist nun eingegeben, der Cursor befindet sich in der nächsten Zeile und die nächsten beiden Befehle können in gleicher Weise eingegeben werden:

```
10 PRINT "INGE"
20 PRINT "UND"
30 PRINT "PETER"
■
```

Da haben wir also das ganze Programm — es ist so gut wie fertig.

FEHLERSUCHE: KORREKTUR EINES PROGRAMMS

Bevor wir unser Programm nun „laufen lassen", sollten wir uns noch einmal **vergewissern**, daß alle Programmteile richtig eingegeben sind. Wir dürfen nie vergessen, daß unser Sony Computer ja nur eine Maschine ist, die ausschließlich die Befehle der Programmiersprache BASIC versteht. Wenn nur ein Buchstabe oder eine Leerstelle nicht stimmen, dann versteht der Computer das Wort falsch oder überhaupt nicht. Das Programm stoppt dann, oder der Computer führt etwas aus, was wir gar nicht wollten. Z.B.:

```
10 PRIMT "INGE"
```

Fehler

Der PRINT-Befehl muß richtig geschrieben werden, sonst versteht der Computer nicht einmal, daß wir ihm überhaupt einen Befehl gegeben haben.

Wenn wir schon jetzt einen Fehler im Programm gefunden haben, sollten wir froh sein: Besser jetzt als später! Das Korrigieren ist im Grunde kein Problem.

Wie der Cursor durch Drücken der Tasten mit Pfeilmarkierung bewegt werden kann, wissen wir ja schon. Der Cursor befindet sich nun am Programmende, und wir müssen ihn erst einmal zur „10" hochholen. Danach bewegen wir ihn um sechs Stellen nach rechts, so daß er sich direkt unter dem fehlerhaften Buchstaben M befindet.

```
10 PRIMT "INGE"
```

Den Cursor an diese Stelle bewegen.

Zum Korrigieren einfach den richtigen Buchstaben eintippen ...

```
10 PRINT "INGE"
```

N drücken.

Nun drücken wir einfach RETURN ... und schon ist der Fehler beseitigt!

```
10 PRINT "INGE"
20 PRINT "UND"
```

Nach Drücken von RETURN befindet sich der Cursor hier.

Die anderen Fehler können wir in gleicher Weise korrigieren. (Wie wir einen Buchstaben oder eine Leerstelle nachträglich einfügen bzw. herausnehmen können, steht in der **Bedienungsanleitung**.) Wenn schließlich alle Fehler korrigiert sind, bewegen wird den Cursor wieder ganz nach unten, so daß sich folgendes Bild ergibt:

```
10 PRINT "INGE"
20 PRINT "UND"
30 PRINT "PETER"
```

Bei diesem einfachen Programm sind uns vielleicht noch gar keine Fehler unterlaufen. Dennoch ist es ganz wichtig, immer alles genau noch einmal zu **überprüfen**. Schon bald werden wir längere Programme mit komplizierteren Befehlen schreiben, und bereits der kleinste Fehler kann zu einem Abbruch des Programmlaufs führen. Fehler unterlaufen **jedem** einmal—selbst den erfahrensten Programmierern bei Sony.

Eins ist ganz sicher: Früher oder später teilt uns der Sony Computer in einer **Fehlermeldung** mit, daß etwas nicht stimmt. Das Programm stoppt dann vorzeitig, oder der Computer macht etwas, was wir gar nicht wollten. In diesem Fall müssen wir die ganze Liste der Befehle noch einmal nach Fehlern absuchen und diese dann korrigieren. Viel besser ist, schon beim Eintippen der Befehle, also vor dem eigentlichen Programmlauf, alles genau zu überprüfen und evtl. zu korrigieren.

Grundsatzregeln beim Programmieren

- **Alle notwendigen Programmschritte** überlegen, damit der Computer das ausführt, was wir wollen.
- Sicherstellen, daß alle Befehle in der **logischen Reihenfolge** angeordnet sind, so daß wir das richtige Resultat erhalten.
- Am **Anfang** jeder Befehlszeile eine **Zeilennummer** eingeben.
- Nach jeder Befehlseingabe **sorgfältig auf Fehlern überprüfen** und evtl. eine Korrektur vornehmen.

PROGRAMMLAUF

Nachdem wir nun die Befehle eingegeben und korrigiert haben, können wir das Programm laufen lassen. Dazu geben wir den **RUN**-Befehl ein. „Run" bedeutet auf Deutsch nichts anderes als „laufen".

```
RUN
```

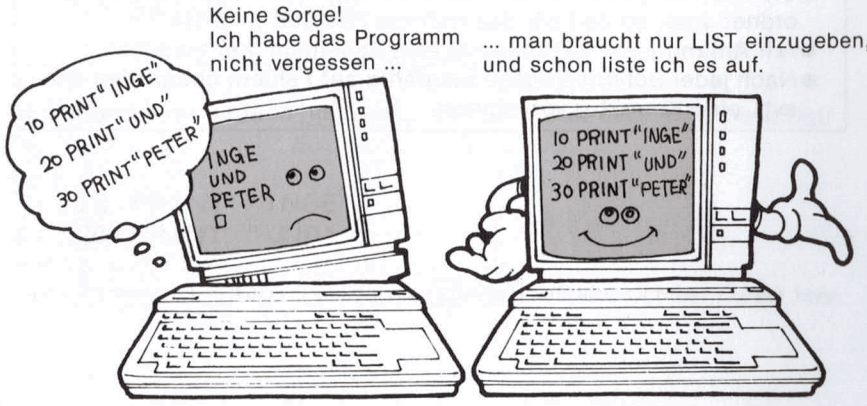
Ist das Programm eingegeben, **und** RETURN gedrückt, so bildet der Computer folgendes auf dem Bildschirm ab:

```
10 PRINT "INGE"
20 PRINT "UND"
30 PRINT "PETER"
RUN
INGE
UND
PETER
Ok
■
```

Zum „Lesen" der Befehle und anschließenden Ausdrucken benötigt der Sony Computer nur Sekundenbruchteile.

Wenn wir nun noch einmal RUN eingeben und RETURN drücken, erhalten wir wieder das gleiche Ergebnis. **Dieses** Programm bleibt immer gleich, es sei denn, wir ändern oder löschen die Befehle. So oft wir es auch laufen lassen, immer wieder erscheint INGE UND PETER auf dem Bildschirm, d.h. der Sony Computer muß unser Programm **gespeichert** haben.

Zum Überprüfen des gespeicherten Programms können wir **LIST** eingeben (und anschließend RETURN drücken). Diesen **LIST-Befehl** werden wir beim Programmieren immer wieder benötigen.



Was unseren Sony Computer so nützlich macht, ist die Tatsache, daß er sich unsere Befehle merken kann. Zu diesem Zweck brauchen wir lediglich **eine Zahl vor dem Befehl** einzugeben.

```
10 PRINT "INGE"
```

Wenn eine Zeilennummer eingegeben wird, handelt es sich um ein Programm.

Dieser Befehl besitzt eine Nummer. Der Computer weiß deshalb sofort, daß es sich um einen Teil eines Programmes handelt und speichert den Befehl. 10 bezeichnen wir als **Zeilennummer**. Die Zeilennummer kann eine beliebige Zahl zwischen 0 und 65529 sein.

Aber wie lange kann sich der Computer unser Programm merken? Eigentlich für immer! Oder bis wir ihm sagen, daß er alles vergessen soll. Der Befehl zum Vergessen ist folgender:

```
NEW
```

Dann wollen wir gleich einmal überprüfen, ob dieser Befehl wirklich funktioniert. Hierzu geben wir einige Befehle mit Nummern ein, die sich unser Computer merkt, und danach dann LIST. Hat der Computer alles aufgelistet? Gut! Dann geben wir NEW ein.

```
LIST
10 PRINT "INGE"
20 PRINT "UND"
30 PRINT "PETER"
Ok
NEW
Ok
■
```

Auf dem Schirm sind zwar noch unsere Befehle zu sehen, der Computer hat sie aber schon vergessen. Wenn wir erneut LIST eingeben, merken wir, daß das Programm gelöscht ist.

Es gibt aber noch zwei andere Arten, den Computer Befehle vergessen zu lassen: Man kann die RESET-Taste drücken oder den Computer einfach ausschalten. **Das sollte man aber nur tun**, wenn man ganz sicher ist, daß man das Programm nicht mehr braucht!

Später werden wir lernen, wie man ein Programm mit einem Recorder sichert, damit man den Computer beruhigt ausschalten und dennoch das Programm nach Wochen oder Jahren wieder verwenden kann. Im Moment jedenfalls vergißt der Computer noch alles, sobald wir ihn ausschalten.

DER COMPUTER ZEICHNET BLINKENDE STERNE

- Weitere BASIC-Befehle
- Löschen des Schirms
- Wir zeichnen Punkte und Linien
- Der Computer wiederholt einen Vorgang
- Verwendung von Variablen
- Verwendung von Zufallszahlen
- Neue COLOR-Befehle

EIN GRAFIK-PROGRAMM

Wir wollen nun einige neue BASIC-Wörter kennenlernen und geben folgendes ein:

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

Auch hierbei handelt es sich wieder um ein **Programm** — also um eine Abfolge von Schritten, die dem Computer sagen, was er tun soll. Wenn das Programm im Memory gespeichert ist, geben wir den RUN-Befehl ein und sehen uns einmal an, was dieses Programm bewirkt.

Sämtliche Buchstaben sind verschwunden, und auf dem Bildschirm sind nur noch vier weiße Punkte zu sehen. Sogar der Cursor ist nicht mehr da, und wenn wir eine Taste drücken, passiert gar nichts auf dem Bildschirm. Der Computer ist jetzt mit der Abarbeitung des Programms beschäftigt. Aber warum dauert das so lange?

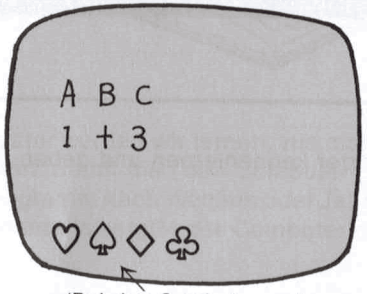
Um das zu verstehen, müssen wir uns das Programm einmal anschauen. Zunächst drücken wir [CTRL] und [STOP], um das Programm zu stoppen. Die Punkte verschwinden dann, und der Cursor ist wieder da. Nun geben wir LIST ein und schauen uns die Programmzeilen im einzelnen an. Zunächst zur Zeile 10.

SCREEN 2

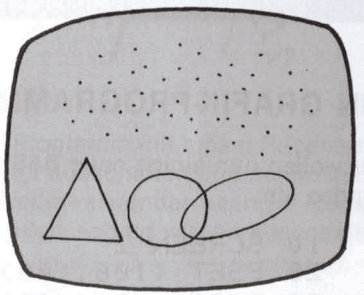
Mit diesem Befehl beginnt der Computer, Grafiken auf dem Bildschirm abzubilden. **Grafiken** können Punkte, Linien, Kreise und andere Figuren oder Muster sein. (Außer diesen Grafiken kann unser Computer auch normale **Zeichen** — d.h. Zahlen, Buchstaben und Symbole—abbilden.)

Mit den SCREEN-Befehlen sagen wir dem Computer, in welcher Form die Informationen auf dem Bildschirm abgebildet werden sollen. Es gibt zwei verschiedene SCREEN-Befehle.

SCREEN 0 oder SCREEN 1 für **Zeichen**



SCREEN 2 für **Grafiken**



(Bei den Symbolen auf unserer Tastatur handelt es sich um Zeichen)

SCREEN 2 bewirkt also, daß unser Computer Grafiken abbildet. Aus diesem Grund sind auch im vorigen Beispiel nach der Eingabe von RUN alle Zeichen vom Bildschirm verschwunden und stattdessen Grafiken erschienen. Mit CTRL und STOP wird der Befehl zum Umschalten auf Grafiken wieder gelöscht, und die normalen Zeichen sind wieder zu sehen. Die erste Zeile unseres Programms haben wir also nun verstanden und können zu den nächsten übergehen.

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

Der PSET-Befehl sagt dem Computer, daß er **Punkte** auf dem Bildschirm abbilden soll, und zwar an den in Klammern angegebenen Stellen.

Wiederholen, wiederholen, wiederholen...
In der letzten Zeile sehen wir einen ganz wichtigen Befehl dieses Programms:

```
60 GOTO 20
```


GOTO bedeutet wörtlich übersetzt „gehe zu", und man nennt diesen Befehl einen Sprungbefehl. In unserem speziellen Fall soll der Computer zur Zeile 20 **zurückspringen** und dort von neuem beginnen. In der Zeile 20 geht es dann weiter mit.

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

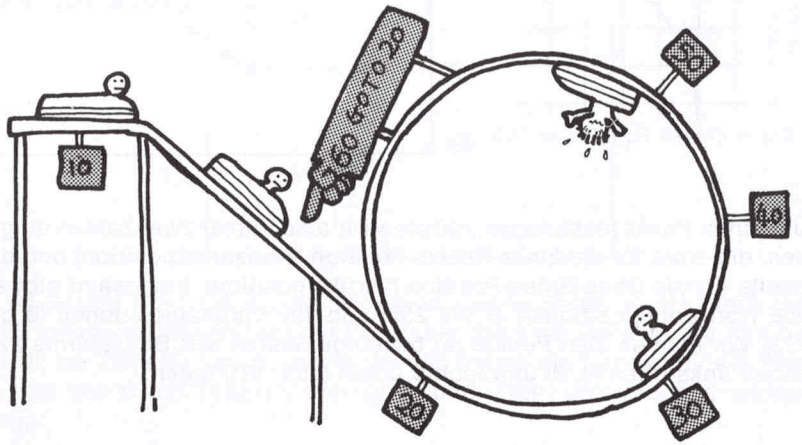
Der Computer wiederholt also das Programm. Wenn er in der Zeile 60 angelangt ist, springt er wieder zur Zeile 20 zurück, und das Programm wird erneut wiederholt usw. usw.

Dieser Programmteil wird endlos wiederholt.

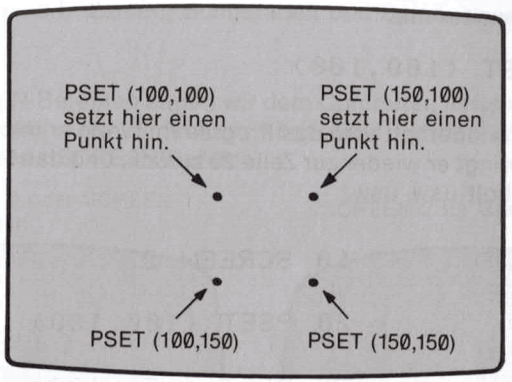
```
10 SCREEN 2
 ↓
20 PSET (100,100)
 ↓
30 PSET (150,100)
 ↓
40 PSET (100,150)
 ↓
50 PSET (150,150)
 ↓
60 GOTO 20
```

Einen solchen Teil, der immer wiederholt wird, nennt man eine **Schleife**—eine beim Programmieren sehr oft verwendete und sehr nützliche Sache. Dabei muß es sich aber nicht unbedingt immer um eine endlose Schleife handeln wie in unserem speziellen Beispiel.

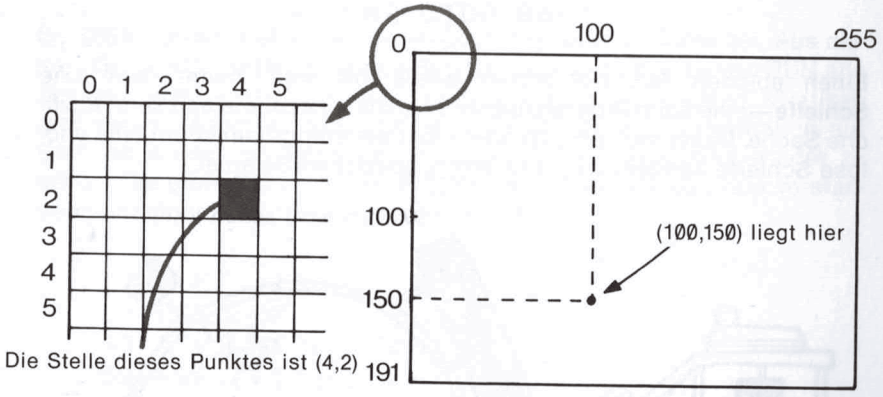
War ich nicht schon einmal hier?



Und wie können wir nun mit dem PSET-Befehl Punkte auf dem Bildschirm abbilden?



Vielleicht haben einige schon gemerkt, welche Bedeutung die Zahlen haben. Bei den beiden Punkten auf der linken Seite ist die erste Zahl 100 und bei den beiden Punkten auf der rechten Seite ist sie 150. Zur Verdeutlichung wollen wir uns einmal einen vergrößerten Bildschirmausschnitt ansehen:



Um einen Punkt festzulegen, müssen wir also immer zwei Zahlen eingeben: die erste für die Links-Rechts-Position (Horizontalposition) und die zweite für die Oben-Unten-Position (Vertikalposition). Insgesamt gibt es 256 Horizontalpositionen (0 bis 255) und 192 Vertikalpositionen (0 bis 191). Wir können also Punkte an beliebige Stellen des Bildschirms zwischen links oben (0, 0) und rechts unten (255, 191) legen.

Grafiken und Variablen

Wir erinnern uns sicher noch daran, daß man Variablen eine beliebige Zahl zuordnen kann. Wenn wir für die Horizontalposition die Variable X und für die Vertikalposition die Variable Y verwenden, sieht unser Befehl wie folgt aus:

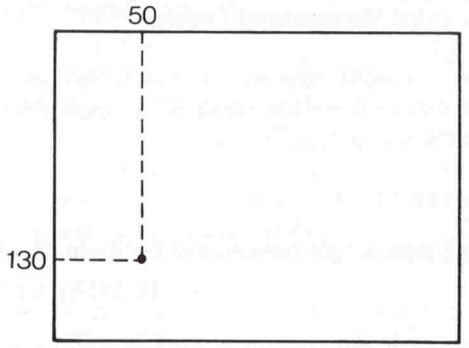
```
PSET (X, Y)
```

X und Y können nun beliebigen Zahlen zugeordnet werden, um die Punkte an bestimmten Stellen abzubilden.

Im folgenden Programm sehen wir, wie man die Variablen zum Abbilden eines Einzelpunktes verwenden kann. Zunächst löschen wir das vorhergehende Programm mit dem NEW-Befehl und geben dann das folgende neue Programm ein:

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

Durch Drücken von RETURN speichern wir das Programm im Speicher ab und geben dann den RUN-Befehl ein. Der Computer beginnt mit der Zeile 10, die den Bildschirm für die **Grafiken** löscht. Durch Zeile 20 werden zwei Variablen Zahlenwerte zugeordnet, und in Zeile 30 wird dann mit diesen Werten ein Punkt auf dem Bildschirm abgebildet.



X ist 50 und Y ist 130, so daß wir einen weißen Punkt an der Stelle (50, 130) erhalten. Danach liest der Computer dann die Zeile 40, die ihm befiehlt, zur Zeile 30 zurückzuspringen. Wir haben also wieder eine Schleife, die nur durch Drücken von CTRL und STOP unterbrochen werden kann.

Aber warum haben wir eigentlich eine Variable verwendet, statt direkt PSET (150, 130) einzugeben? Und warum haben wir den Computer mit Zeile 40 in eine Endlosschleife gebracht? Zum kurzzeitigen Abbilden eines Einzelpunktes auf dem Bildschirm wären diese Schritte ja doch eigentlich gar nicht notwendig. Wenn wir jedoch eine Menge Punkte abbilden wollen, können wir dies mit der beschriebenen Methode durch ein sehr kurzes Programm zustande bringen. Genau das wollen wir uns als nächstes, nach einigen kurzen Hinweisen zu Satzzeichen und zur Schreibweise, etwas genauer anschauen.

Hinweis: Vorsicht mit den Satzzeichen!

Das Programm, das wir gerade erstellt haben, besitzt drei verschiedene **Satzzeichen**—ein **,** (Komma) und einen **:** (Doppelpunkt) und **( )** (Klammern). Auch hierbei handelt es sich um **Zeichen**, die genau wie die Buchstaben von Befehlen **an den richtigen Stellen eingegeben werden müssen.** Wir erinnern uns, daß unser Sony Computer einen Befehl nicht verstehen kann, wenn ein Tippfehler vorkommt. Genau das gleiche gilt auch für die Satzzeichen.

Jedes Satzzeichen hat in BASIC eine spezielle Bedeutung. Das Komma beispielsweise zeigt an, daß eine Zahl zu Ende ist und eine andere beginnt. Ein Doppelpunkt dagegen bedeutet, daß ein Befehl zu Ende ist und in der gleichen Zeile noch ein weiterer Befehl folgt. Klammern werden benötigt, wenn zwei oder mehr Zahlen bzw. zwei oder mehr Variablen zusammen verwendet werden und müssen **immer paarweise** verwendet werden, d.h. links befindet sich „(" und dann rechts „)". Im späteren Verlauf des Buchs werden wir dann noch andere Satzzeichen kennenlernen, die bei BASIC-Befehlen Verwendung finden.

Wie bereits erwähnt, macht jeder einmal Tippfehler. Der Sony Computer kann im allgemeinen den Befehl dann nicht verstehen und teilt dies durch eine Fehlermeldung mit. Z.B.:

```
Syntax error in 30
```

Dies bedeutet, daß sich möglicherweise in der Zeile 30 ein Tippfehler befindet.

ZUFALLSZAHLEN

Wir löschen wieder den Speicher des Computers mit dem NEW-Befehl und geben dann folgendes Programm ein:

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

Wenn wir nun das Programm mit RUN starten, füllt sich der Bildschirm mit mehr und mehr Punkten, bis wir das Programm mit CTRL und STOP abbrechen.

Was ist denn mit meinem Gesicht passiert?

Um dies zu verstehen, müssen wir uns die beiden neuen Zeilen des Programms anschauen.

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

Es ist leicht einzusehen, daß diese beiden Zeilen den Variablen X und Y bestimmte Werte zuteilen. Aber welche Werte sind das nun? Was bewirkt der RND(1)-Befehl? RND ist eine Abkürzung des englischen Wortes **random**, und bedeutet soviel wie „Zufall". RND ist eine der **Funktionen** der Programmiersprache BASIC. Um diese **Zufallszahlen-Funktion** zu verstehen, probieren wir einmal folgenden Befehl aus.

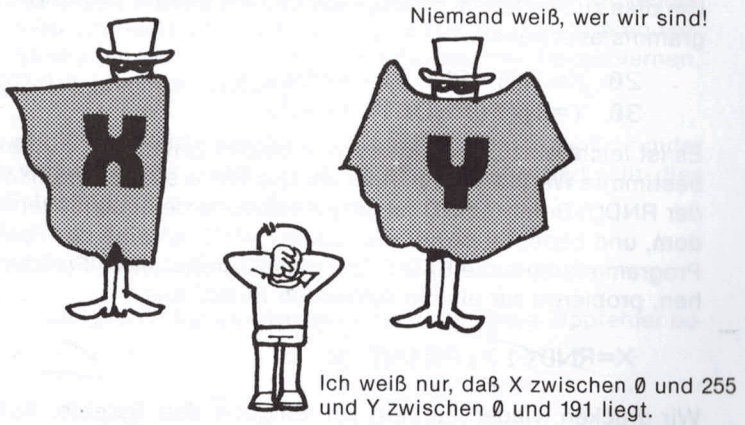
```
X=RND(1):PRINT X
```

Wir drücken wieder RETURN zur Eingabe des Befehls. Auf dem Bildschirm erscheint eine Zahl, die kleiner als 1 ist und 14 Stellen hinter dem Komma besitzt. Wenn wir genau hinsehen, merken wir, daß der Computer aber kein Dezimalkomma, sondern einen **Dezimalpunkt** verwendet. Z.B. könnte folgende Zahl zu sehen sein:

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

Wenn wir das gleiche noch einmal probieren, werden wir sehen, daß sich die Zahl geändert hat. Unser Computer kennt nämlich sehr viele solcher Zufallszahlen.

Nun zurück zur Zeile 20 unseres Programms: (RND(1) * 256) bedeutet, daß die Zufallszahlen mit 256 multipliziert werden. Um die Punkte auf dem Bildschirm festzulegen, benötigen wir **ganze** Zahlen, also beispielsweise 1, 30 oder 192. Wenn wir jedoch die Zufallszahlen, die uns der Computer liefert, mit 256 multiplizieren, erhalten wir keine ganzen Zahlen, sondern im allgemeinen nur Bruchzahlen. Aus diesem Grund steht am Anfang **INT**. Dies kommt von **integer** und bedeutet auf Deutsch „ganze Zahl"; d.h. die Stellen nach dem Dezimalpunkt werden einfach abgeschnitten. Jedesmal, wenn der Computer den Befehl X=INT(RND(1) * 256) liest, wird der Wert von X auf diese Weise abgerundet, und wir erhalten irgendeine Zahl zwischen 0 und 255 — wobei wir diese Zahl jedoch niemals voraussagen können. Genau das gleiche bewirkt Zeile 40: Wir erhalten für Y einen zufälligen Wert zwischen 0 und 191.



Auch die nächste Zeile

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

können wir nun leicht verstehen, selbst wenn die Werte von X und Y nicht bekannt sind. Der Computer wird uns diese Werte auch nicht verraten, da wir ja keinen PRINT-Befehl eingegeben haben. Stattdessen wird er immer mehr Punkte an den Stellen der berechneten vertikalen und horizontalen Positions-Zahlen abbilden.

Aber warum kommen immer mehr Punkte auf den Bildschirm? Dies liegt daran, daß sich der Computer in einer Schleife befindet, wie wir in der nächsten Zeile sehen.

```
50 GOTO 20
```

Hierdurch springt der Computer zurück zu

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

und wir erhalten wieder neue Werte für X und Y, so daß durch

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

ein neuer Punkt an einer neuen Stelle abgebildet wird. Dies wird nun immer weiter fortgesetzt, so daß sich die Punkte vermehren.

Programme: Zwei Befehlsarten

Wie wir bereits wissen, ist ein Programm **eine Abfolge von Befehlen in einer festen, numerierten Reihenfolge**, die in den Speicher des Computers eingegeben sind. Ein Programm kann hundert, tausend oder noch mehr Zeilen bzw. Befehle haben. Normalerweise sind aber gar nicht so viele Zeilen notwendig.

Wie wir gerade gesehen haben, können wir die Abbildung von tausend oder noch mehr Punkten mit nur fünf Programmzeilen erreichen.

Eine dieser fünf Befehlszeilen—und zwar GOTO 20 — hat im Grunde gar nichts bewirkt, außer daß dem Computer gesagt wurde, **wohin er als nächstes springen soll**. Und gerade dieser Zeile verdanken wir es, daß unser kurzes Programm so viele Punkte abbilden konnte—da sie ja eine endlose Schleife einleitete.

Es gibt also zwei unterschiedliche Befehlsarten:

1. **Befehle, die dem Computer sagen, was er tun soll, wie z.B.:**

   PRINT PSET WIDTH COLOR usw.

2. **Befehle, die die Abfolge des Programms ändern**

GOTO IF-THEN FOR-NEXT usw.

Von der zweiten Befehlsart gibt es nur wenige, und diese werden wir später im Buch kennenlernen. Sie sind vielleicht die interessantesten Hilfsmittel beim Programmieren: Genau wie der GOTO-Befehl, der eine Schleife einleitete, ermöglichen auch die anderen, **kleine, aber leistungsstarke** Programme zu erstellen.

WIR BRINGEN FARBE INS PROGRAMM

Noch interessanter wird die Sache, wenn wir den Computer die Punkte farbig abbilden lassen. Dazu lassen wir zunächst wieder mit dem LIST-Befehl das im Memory gespeicherte Programm auflisten. (Wenn der Computer zuvor einmal ausgeschaltet wurde, müssen wir allerdings das ganze Programm erneut eingeben.)

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

Nun ändern wir die Zeile 40 zu: 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
```

Wir bewegen den Cursor an diese Position und geben dann **,2** ein. (Das Komma darf auf keinen Fall vergessen werden.) Nun drücken wir RETURN einmal zum Eingeben des Programms und dann gleich noch einmal, um den Cursor zum Programmende zu bewegen.

Wenn wir nun das Programm mit dem RUN-Befehl starten, sind die Punkte nicht mehr weiß, sondern mittelgrün, da durch den Code 2 diese Farbe gewählt wird. Man kann natürlich auch eine beliebige andere Farbe aus der Farbentabelle von Seite 14 auswählen. Wir können die Farbe sogar zu einer Variablen machen und den Computer die Farbe der Punkte selbständig ändern lassen.

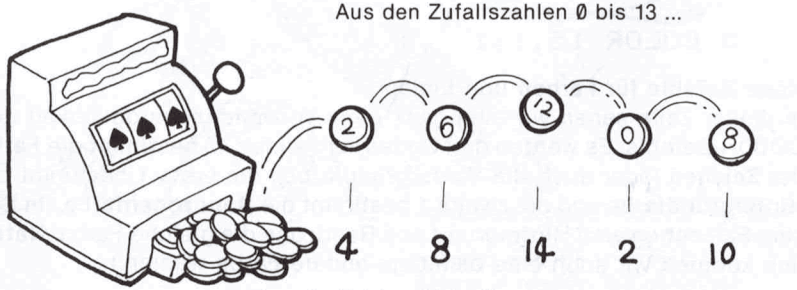
Hierzu gehen wir zurück zur Zeile 40 und tippen statt des Farbcodes die Variable C ein.

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

Der Variablen C muß nun natürlich noch ein Wert, der dann die Farbe bestimmt, zugeordnet werden. Um das ganze noch spannender zu gestalten, können wir auch hier wieder den Computer einen Zufallswert auswählen lassen. Der Befehl dazu ist folgender: (Aber bitte jetzt noch nicht eingeben.)

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

Was bedeutet dies? Es ist ein Befehl für Zufallszahlen genau wie wir ihn schon früher für X und Y verwendet haben, allerdings mit einem Zusatz am Ende. Es wird eine Zufallszahl zwischen 0 und 13 gewählt und dann 2 hinzuaddiert (+2). C erhält also einen Wert zwischen 2 und 15. Insgesamt stehen uns aber 16 Farben — von 0 bis 15 — zur Verfügung. Sicher werden jetzt einige fragen, warum wir die Farben 0 und 1 auslassen? Das ist ganz einfach: 0 bedeutet transparent, und transparente Punkte kann man natürlich nicht sehen, und 1 bedeutet schwarz, was auf einem schwarzen Hintergrund ebenfalls nicht zu sehen ist.



Wo muß aber nun diese Zeile in das Programm eingefügt werden? Natürlich vor dem Befehl zum Abbilden der Punkte, also vor Zeile 40. Wir geben deshalb der neuen Zeile die Nummer 35. Nun wissen wir auch endlich, warum wir für die Zeilennummern bisher immer 10, 20, 30...gewählt haben. Wir haben nämlich dadurch nun noch Platz, nachträglich neue Befehle einzufügen.

```
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. Die Numerierung stimmt zwar, aber wir haben die Befehle nicht der Reihe nach eingetippt. Ob unser Computer das Programm dennoch versteht? Natürlich versteht er es! Er kann ja schließlich zählen und die Zeilen selbständig nach ihrer Numerierung ordnen. Das kann man auch leicht überprüfen, indem man den LIST-Befehl erneut eingibt.

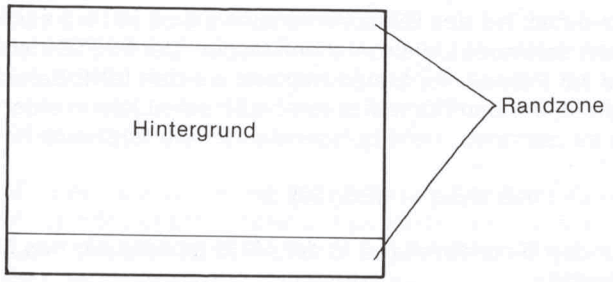
```
LIST
10 SCREEN 2
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
35 C=INT(RND(1)*14)+2
40 PSET (X,Y),C
50 GOTO 20
```

Nun ist also alles vorbereitet, um unsere Punkte in **Sterne** zu verwandeln. Sterne sieht man natürlich nur bei Nacht, und deshalb müssen wir den Bildschirm dunkel machen. Hierzu ist noch eine neue Zeile erforderlich:

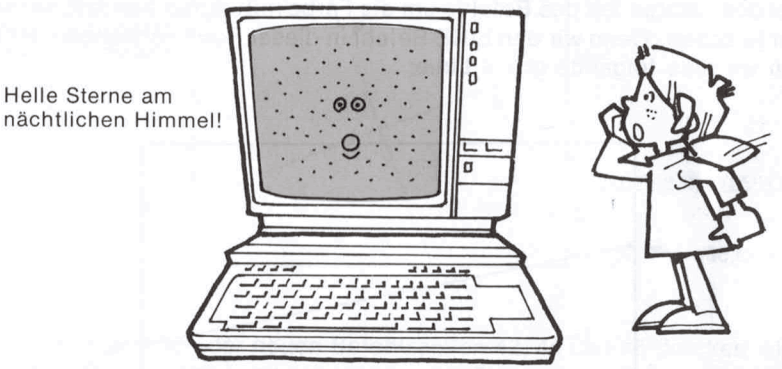
```
5 COLOR 15,1,1
```

Neue Befehle für Farben und Linien

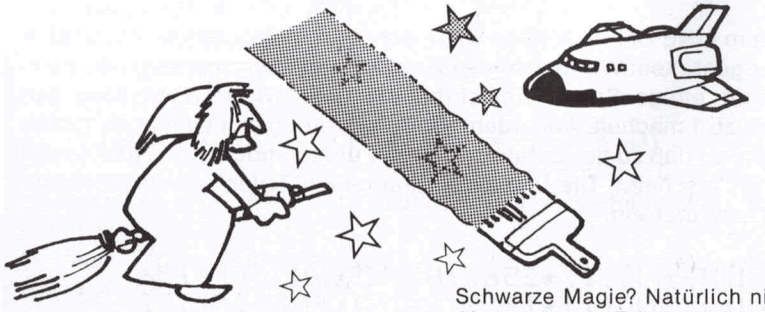
In dieser Zeile sehen wir eine ganz neue Verwendungsmöglichkeit des COLOR-Befehls. Es werden drei Codes verwendet: 15 bestimmt die Farbe der **Zeichen** (oder auch die Vordergrundfarbe), die erste 1 bestimmt die **Hintergrundfarbe** und die zweite 1 bestimmt die **Randzonenfarbe**. (In diesem Fall haben also Hintergrund und Randzone die gleiche Farbe. Natürlich könnten wir auch eine beliebige andere Farbe wählen.)



Nun kann es also losgehen: RUN ...



Aber wir sind nocht fertig! Mit einer kleinen Programmerweiterung werden die Sterne noch natürlicher aussehen. Zunächst wollen wir einmal verhindern, daß eine ganze Galaxis auf unserem Bildschirm erscheint: Wie immer drücken wir CTRL und STOP.



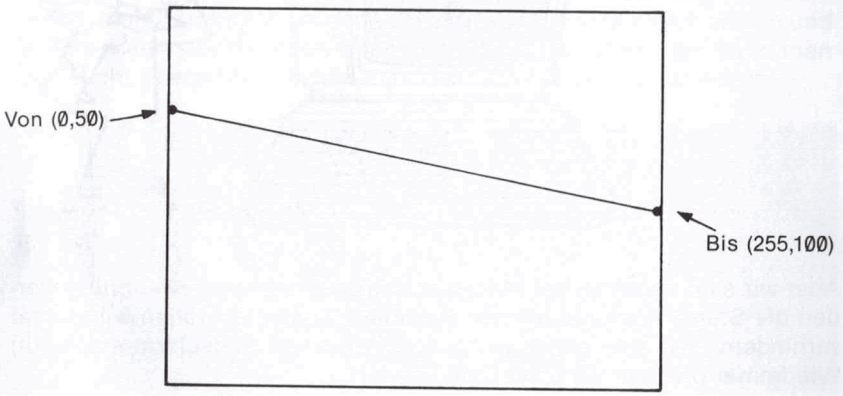
Schwarze Magie? Natürlich nicht! Aber wenn ein Teil des Bildschirmes schwarz gefärbt wird, sind die Sterne in diesem Teil natürlich nicht sichtbar.

Um einen Teil des Bildschirms schwarz zu färben, können wir den Computer **schwarze Linien** zeichnen lassen. Der PSET-Befehl war bekanntlich nur für Punkte; für Linien müssen wir den **LINE**-Befehl verwenden. Ein Befehl, der den Computer veranlaßt, eine Linie in einer bestimmten Farbe zu zeichnen, sieht beispielsweise wie folgt aus:

```
LINE (Ø,5Ø) – (255,1ØØ), 2
```

Für den Bindestrich [–] in der Mitte müssen wir das Minuszeichen verwenden.

Der Bindestrich bedeutet „von/bis", d.h. mit diesem Befehl wird eine Linie **von** der Position (Ø,5Ø) **bis** zur Position (255, 1ØØ) gezeichnet. Daß es sich bei dem letzten Teil des Befehls um die Farbe mittelgrün handelt, wissen wir ja schon. Wenn wir den LINE-Befehl in dieser Form verwenden, erhalten wir also folgende grüne Linie:



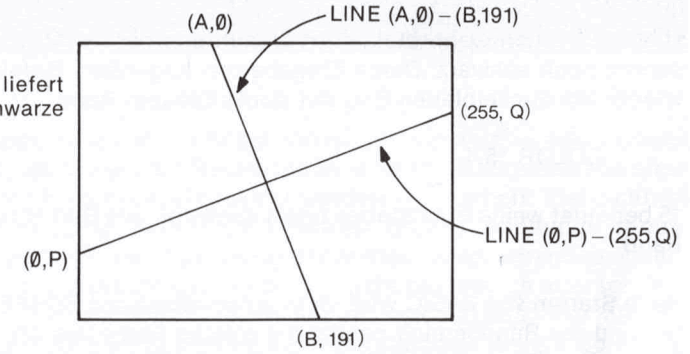
Mit diesem LINE-Befehl können wir unser „Sterne-Programm" noch interessanter gestalten: Wir lassen den Computer eine schwarze Linie zeichnen, damit einige Sterne unsichtbar werden. Wir müssen also den Farbcode zu 1 machen. Außerdem werden wir auch hier wieder Variablen einsetzen, so daß zu der Zufallsserie von Punkten noch eine Zufallsserie von Linien erscheint. Die folgenden drei neuen Zeilen geben wir hierzu in den Computer ein:

```
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,Ø)-(B,191),1:LINE (Ø,P)-(255,Q),1
```


Jeder dieser Befehle ist so lang, daß er mehr als eine Zeile auf dem Bildschirm ausfüllt; dies ist jedoch überhaupt kein Problem. Der Computer schreibt automatisch solche Befehle über zwei Zeilen, behandelt sie jedoch nach wie vor als eine Zeile, da ja nur jeweils **eine Zeilennummer** (42, 44 bzw. 46) eingegeben wurde.

In Zeile Nummer 46 haben wir zwei LINE-Befehle mit vier neuen Variablen. In den Zeilen 42 und 44 werden mit den uns schon bekannten Befehlen wieder den neuen Variablen A, B, P und Q Zufallszahlen zugeteilt. Und so sieht ein Paar der schwarzen Zufallslinien aus:

Befehlszeile 46 liefert unsichtbare schwarze Linien.



Nach dem Eintippen der neuen Befehlszeilen 42, 44 und 46 drücken wir jeweils wieder RETURN und geben dann LIST ein, damit das ganze Programm zu sehen ist.

```
LIST
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
```

Die neuen Befehlszeilen müssen wir vor den GOTO-Befehl — also vor Zeile 50—einfügen, damit sie sich genau wie die Befehlszeilen für die Sterne innerhalb der Endlosschleife befinden. Jedesmal wenn der Computer die Befehlszeilen 20 bis 46 abarbeitet, bildet er zufällig verteilte neue Punkte und Linien ab. Und wenn die Punkte zufällig auf einer schwarzen Linie liegen, sind sie unsichtbar. Nun geben wir also wieder den RUN-Befehl ein ... und siehe da, die Sterne **blinken**!

Wenn wir die Linien wirklich sehen wollen, um den ganzen Vorgang besser verstehen zu können, müssen wir im Befehl der Zeile 46 den Farbcode von 1 zu 2 ändern. Es erscheinen dann grüne Linien auf dem Bildschirm.

HINWEIS: Auch wenn dieses Programm gestoppt wird, bleibt der Bildschirm noch schwarz. Durch Eingabe von folgendem Befehl erhält man wieder ein dunkelblaues Bild mit dunkelblauem Rand:

```
COLOR 15,4
```

15 bedeutet weiße Buchstaben und 4 dunkelblaues Bild mit dunkelblauem Rand.

Beim Starten von BASIC wird stets automatisch auf SCREEN 0 geschaltet, und der Randbereich besitzt die gleiche Farbe wie der Hintergrund. Sollen Randbereich und Hintergrund unterschiedlich gefärbt sein, so müssen wir zunächst SCREEN 1 und dann beispielsweise COLOR 15, 4, 7 eingeben.
Zum Zurückschalten in den Originalzustand können wir später jederzeit wieder SCREEN 0 eingeben.

SICHERN DES PROGRAMMS

- Anschluß des Computers an einen Cassettenrecorder
- Überspielen eines Programms auf den Cassettenrecorder
- Prüfen, ob die Überspielung richtig ausgeführt wurde
- Laden eines Programms vom Cassettenrecorder in den Computer

Vielleicht haben einige schon Bilder von Großcomputern — also Computern, die beispielsweise eine Rakete leiten, eine Produktionsanlage steuern oder in der Forschung eingesetzt werden — gesehen. Vielleicht ist einigen auch etwas aufgefallen, was wie ein großes Tonbandgerät aussieht. Es handelt sich dabei um den sogenannten „Magnetbandspeicher" des Computers und dieser sieht in der Tat nicht nur ähnlich aus, sondern hat auch eine ganz ähnliche Funktion wie ein Tonbandgerät. Der Magnetbandspeicher dient zum Speichern der Informationen, die ein Computer benötigt.

So können beispielsweise auch Programme — also Anweisungen an die Maschine wie z.B. die uns schon bekannten Befehle PSET, SCREEN, PRINT — gespeichert werden. Weiterhin lassen sich aber auch **Daten** wie die Punktzahl bei einem Spiel, die Antworten auf Probleme oder wissenschaftliche Formeln abspeichern.

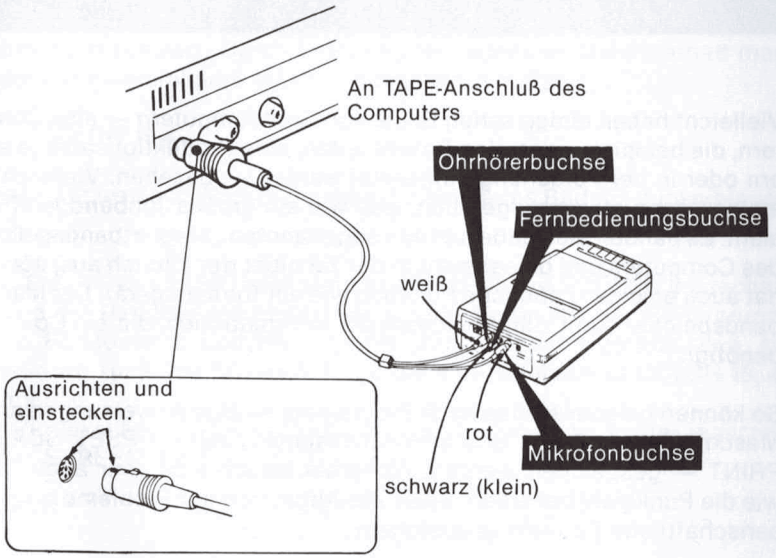
Die Computertechnik hat sich unglaublich schnell entwickelt und Aufgaben, für die früher ein Großcomputer erforderlich war, können heutzutage problemlos von kleineren Geräten wie dem Sony Computer bewältigt werden. Obwohl unser Sony Computer so leicht ist, daß ihn jedes Kind heben kann, ist er sogar noch schneller und intelligenter als viele der alten Computer, die etliche zehntausend DM kosteten und ganze Räume ausfüllten.

Unser Sony Computer kann sogar Programme und andere Informationen auf Band abspeichern. Ein Magnetbandspeicher, wie er für Großcomputer verwendet wird, wäre natürlich viel zu groß für uns. Unser kleiner Sony Computer begnügt sich schon mit einem ganz normalen Cassettenrecorder, der normalerweise für Musik oder Diktate verwendet wird.

ANSCHLUSS EINES CASSETTENRECORDERS

Zum Anschließen brauchen wir lediglich das beim Sony Computer mitgelieferte Spezialkabel zu verwenden. Der Stecker mit den Stiften ist hierzu in den mit TAPE gekennzeichneten Anschluß auf der Rückseite der Tastatur einzustecken.

Am anderen Kabelende sind drei Stecker, ein roter, ein weißer und ein schwarzer. Den roten Stecker stecken wir in die Mikrofonbuchse des Cassettenrecorders und den weißen in die Ohrhörerbuchse. Wenn der Cassettenrecorder auch noch eine Fernbedienungsbuchse besitzt, so stecken wir den schwarzen Stecker in diese Buchse. (Falls nicht, bleibt der schwarze Stecker frei.)



Nun können sich Computer und Cassettenrecorder miteinander „unterhalten".

WIR LASSEN DEN COMPUTER MIT DEM CASSETTENRECORDER „SPRECHEN"

Wenn wir folgenden Befehl eingeben, teilt der Computer dem Cassettenrecorder seinen Speicherinhalt mit:

```
CSAVE "Dateiname"
```

Im CSAVE-Befehl bedeutet C Cassette, SAVE heißt soviel wie sichern und eine „Datei" ist eine bestimmte Datenansammlung: Das im Speicher des Computers befindliche Programm wird also auf Cassette gespeichert und damit gesichert. Dabei können wir uns einen beliebigen "Dateinamen" für unser Programm ausdenken. (Die beiden Anführungszeichen aber nicht vergessen!) Dieser Name dient lediglich dazu, daß der Cassettenrecorder, der Computer und auch wir selbst die einzelnen Dateien nicht durcheinander bringen.

Ein **Dateiname** darf bis zu sechs Zeichen besitzen. Dabei muß das **erste** Zeichen ein **Buchstabe** sein, und alle anderen können sich beliebig aus Buchstaben, Zahlen und Grafiken zusammensetzen.

STERN würde sich beispielsweise gut für das Programm aus dem letzten Kapitel eignen. Es besteht aus fünf Zeichen, beginnt mit einem Buchstaben, und man kann es sich gut merken, da das Programm ja Sterne auf dem Bildschirm hervorzaubert.

Zum Sichern geben wir ein:

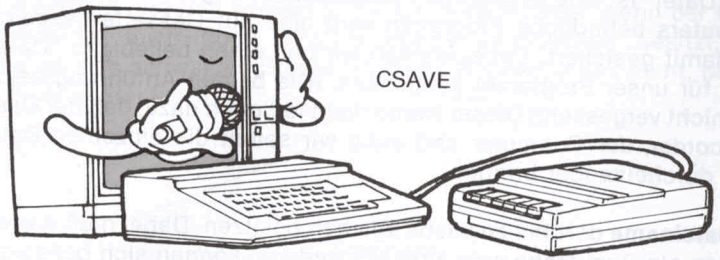
```
CSAVE "STERN"
```

Aber bitte die RETURN-Taste noch nicht drücken, da der Computer sonst sofort beginnt, die zu speichernden Informationen an den Cassettenrecorder zu schicken. Zuerst müssen wir uns noch einmal vergewissern, daß der Cassettenrecorder auch wirklich bereit ist.

Wenn der Cassettenrecorder eine Fernbedienungsbuchse besitzt und das Kabel daran angeschlossen ist, müssen wir den Recorder auf Aufnahme schalten. Wie wir sehen werden, läuft das Band jedoch nicht sofort los, da der Computer die „Bedienung" unseres Cassettenrecorders in die Hand nimmt. Erst wenn wir RETURN drücken, läuft das Band los, das Programm wird aufgezeichnet, und am Ende stoppt das Band automatisch wieder. Am Ende der Aufzeichnung erscheint folgendes auf dem Bildschirm:

```
CSAVE "STERN"
Ok
■
```

Wenn der Cassettenrecorder keinen Fernbedienungsanschluß besitzt, müssen wir zum Starten des Bandlaufs **die Aufnahmetaste drücken**. Nun **warten** wir einige Sekunden, bis sich die Bandgeschwindigkeit stabilisiert hat, und drücken dann [RETURN]. Sobald nun Ok auf dem Bildschirm erscheint, **drücken wir die Stopptaste** des Recorders, und sind damit praktisch schon fertig.



CSAVE

Das Programm befindet sich nun immer noch im Speicher des Computers und — wenn wir alles richtig gemacht haben — zusätzlich noch auf der Cassette.

IST AUCH WIRKLICH ALLES RICHTIG GESICHERT WORDEN?

Bevor wir irgendetwas anderes machen, sollten wir **immer überprüfen**, ob der **Cassettenrecorder** auch alles richtig abgespeichert hat. Der Computer kann die Überprüfung selbst ausführen, indem er seinen Speicherinhalt mit der Cassetten-Aufzeichnung vergleicht. Wir müssen hierzu zunächst das Kabel vom Fernbedienungsanschluß **abziehen** und dann das Band zum Anfang des gerade gesicherten Programms **spulen**. Nun **stellen wir den Lautstärkeregler** des Cassettenrecorders ungefähr in die Mittelstellung, **schließen** den Fernbedienungsstecker **wieder an** und geben folgenden Befehl ein:

```
CLOAD? "STERN"
```

Wenn kein Fernbedienungsanschluß vorgenommen wurde, müssen wir [RETURN] drücken, **bevor** wir den Recorder auf Wiedergabe schalten. Ist dagegen ein Fernbedienungsanschluß vorgenommen, schalten wir den Recorder zuerst auf Wiedergabe. Sobald wir [RETURN] drücken, startet das Band dann automatisch. Wenn der Anfang des Programms vom Computer gelesen wird, erscheint folgendes auf dem Bildschirm:

```
Found:STERN
```


Während der Computer die Bandwiedergabe steuert, vergleicht er gleichzeitig das aufgezeichnete Programm Punkt für Punkt mit dem Programm in seinem Speicher und wenn alles in Ordnung ist, erscheint Ok auf dem Bildschirm. Wenn keine Fernbedienung vorhanden ist, müssen wir danach das Band wieder per Hand stoppen.

Diese Überprüfung ist sehr wichtig, da manchmal ein Problem vorhanden sein kann — und schließlich wollen wir nicht das einmal geschriebene Programm verlieren. Wenn die „Found"-Meldung nicht auf dem Bildschirm erscheint, wurde das Band vielleicht nicht weit genug zurückgespult.

Erscheint **kein** Ok-Zeichen, so können wir versuchsweise einmal am Recorder die Lautstärke erhöhen und das ganze noch einmal probieren. Klappt es dann wieder nicht, so liegt eine elektrische Störung vor, oder das Programm ist nicht richtig auf Band aufgezeichnet worden. In diesem Fall müssen wir noch einmal alle Anschlüsse überprüfen, bevor wir dann erneut den Befehl CSAFE "STERN" eingeben.

Wenn nach dem Überprüfen ein Ok erscheint, können wir sicher sein, daß das Programm richtig auf Cassette gespeichert wurde. Wir können die Cassette dann beschriften und für eine spätere Verwendung des Programms irgendwo aufbewahren.

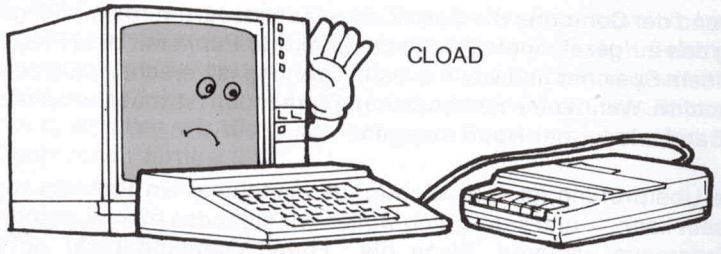
Wenn das Programm richtig auf Cassette gespeichert wurde, können wir NEW eingeben oder die [RESET]-Taste drücken. Der **Speicher des Computers** wird dann gelöscht — aber das ist nun kein Problem mehr, da wir ja noch die **externe Speicherung** auf Band haben.

LADEN EINES PROGRAMMS

Das Zurückholen des Programms von der Cassette in den Speicher des Computers bezeichnet man als „Laden des Programms". Und damit werden wir uns nun im folgenden beschäftigen. Der Vorgang ist ähnlich wie die Überprüfung, die wir gerade durchgeführt haben. Zunächst vergewissern wir uns einmal, daß der Recorder auf **Wiedergabe** geschaltet ist und die eingelegte Cassette kurz vor den Anfang des aufgezeichneten Programms gespult ist. Ist alles in Ordnung, geben wir ein:

```
CLOAD "STERN"
```

Diesmal erscheint kein Fragezeichen. (Natürlich müssen wir wieder darauf achten, den Dateinamen richtig einzugeben.)



Wenn der Computer alles richtig aufgenommen hat, erscheint auf dem Bildschirm:

```
CLOAD "STERN"
Found:STERN
Ok
■
```

Nun brauchen wir nur noch den Cassettenrecorder zu stoppen — und schon können wir durch Eingabe von RUN das Programm auf unserem Computer laufen lassen.

Es kann jedoch auch vorkommen, daß der Computer folgendes anzeigt:

```
Device I/O error
```

I/O bedeutet **Input/Output** — also Eingang/Ausgang. Der Computer weist uns mit dieser Mitteilung darauf hin, daß irgendetwas mit dem Anschluß des Cassettenrecorders nicht in Ordnung ist. Wir können dann beispielsweise die Lautstärke etwas erhöhen und erneut versuchen zu laden.

ENTSCHEIDUNGEN UND SPIELE

- Unser Computer fällt Entscheidungen
- Programmierung einer Bedingungsgleichung
- Wir programmieren ein Spiel
- Das Programm wird vereinfacht
- Wir verfeinern das Programm

Wir erinnern uns sicher noch an Bello, unseren freundlichen Hund, der viele einfache Kunststücke wie „Sitz!", „Bring!" und „Gib Pfötchen!" beherrscht. Auch die Kunststücke, die unser Sony Computer ausgeführt hat, sind im Grunde noch ziemlich einfach: „PRINT", „GOTO", „PSET" usw.

Der Superhund konnte schon einiges mehr bieten. Wir erinnern uns sicher an die Sache mit den schwarzen und weißen Bällen. Das Kunststück, das er uns vorführte, hing davon ab, was für einen Ball er gefunden hatte. Mit anderen Worten, der Superhund konnte selbständig **Entscheidungen fällen**. Wir können uns schon denken, daß auch unser Sony Computer diese Fähigkeit besitzt und werden im Anschluß nun „Super-Programme" schreiben, in denen auch Entscheidungen gefällt werden müssen.

Wir beginnen wieder, indem wir das letzte Programm mit dem NEW-Befehl löschen und dann folgendes eingeben:

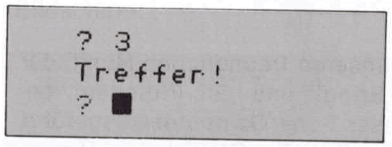
```
10 A=INT(RND(1)*5)+1
20 INPUT B
30 IF A=B THEN GOTO 50
40 GOTO 10
50 PRINT "Treffer!"
60 GOTO 10
```

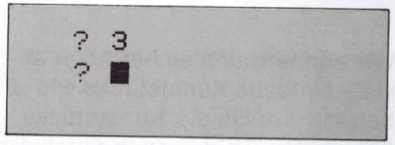
Wenn das richtig eingetippt wurde, geben wir den RUN-Befehl und drücken dann RETURN.

```
RUN
? ■
```

Wieder hat uns der Computer ein Fragezeichen abgebildet, diesmal steht der Cursor jedoch in der gleichen Zeile und nicht in der nächsten. Das bedeutet, daß er auf die **Eingabe** wartet, die wir ihm durch den INPUT-Befehl in Zeile 20 angekündigt haben. In diesem Programm können wir durch einfaches Drücken einer Zahlentaste eine beliebige Zahl zwischen 1 und 5 eingeben. Dann wollen wir dem Computer einmal die Zahl 3 eingeben. (Was passiert wohl, wenn wir einen Buchstaben oder eine Grafik eingeben?)

Wir drücken also [3] und [RETURN]. Welche Anzeige erscheint?





Beide Male haben wir am Ende ein Fragezeichen, das uns sagt, daß der Computer auf die Eingabe einer weiteren Zahl wartet. Dieses Spiel können wir beliebig fortsetzen.—Es ist in der Tat ein richtiges Spiel: Wenn wir die richtige Zahl geraten haben, erscheint „Treffer!" — im anderen Fall erscheint ein „?". Wie oft wird es uns wohl gelingen, „Treffer!" zu erzielen? (Die Zufallsgesetze besagen, daß auf fünf Versuche im Schnitt ein Treffer kommt bzw. auf zehn Versuche zwei Treffer oder auf hundert Versuche zwanzig Treffer kommen.) Mit Sicherheit kann man sagen, daß der Computer eine längere Ausdauer hat als wir. Er wird **niemals** müde, und deshalb müssen wir irgendwann einmal [CTRL] und [STOP] drücken.

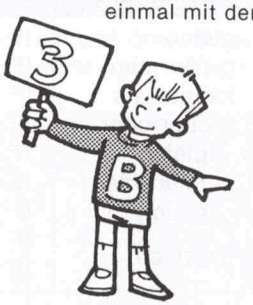
Was passiert nun in dem Programm eigentlich? In der ersten Zeile steht ein alter Bekannter — der **Zufallszahlen-Befehl**. Jedesmal, wenn das Programm durch die von Zeile 60 eingeleitete Schleife läuft, wählt es zufällig eine Zahl zwischen 1 und 5 aus und ordnet diese der Variablen A zu.

In Zeile 20 finden wir dagegen etwas Neues:

```
20 INPUT B
```

B ist die zweite Variable in diesem Programm, und die Zuordnung einer Zahl erfolgt durch Drücken einer Zahlentaste an der Tastatur. Ist dies erfolgt, geht der Computer zur nächsten Zeile.

A ist irgendeine Zahl zwischen 1 und 5.



Als erstes versuchen wir es einmal mit der Zahl 3 ...

TREFFER ODER NICHT — DER COMPUTER ENTSCHEIDET ES

In Zeile 30 werden wir wieder an unseren Superhund erinnert. Die Anweisung beginnt mit dem Wort IF, und das bedeutet auf Deutsch WENN. Hier wird also eine Entscheidung gefällt.

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

Auf IF folgt **immer** THEN, was soviel wie DANN bedeutet. Dieser Befehl wird etwa wie folgt eingesetzt: „WENN du Hunger hast, DANN esse etwas!" oder „WENN du einen Sony Computer hast, DANN spiele mit ihm!"

IF [Bedingung] THEN [mache etwas!]

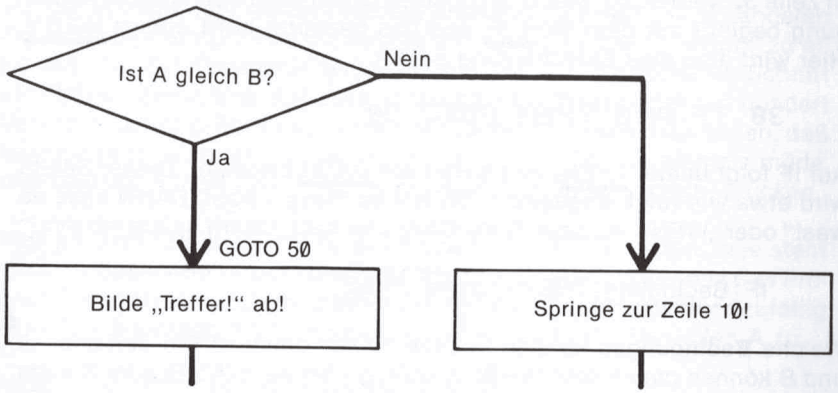
Welche **Bedingungen** können in Zeile 30 vorkommen? Die Variablen A und B können gleich sein (A=B), A kann größer sein (A>B) oder B kann größer sein (A<B).

„Mache etwas!" kann ein Befehl sein. In diesem Fall ist es der Befehl GOTO 50.

Bei BASIC gibt es sechs **Bedingungsgleichungen**, und einige können auf zwei verschiedene Arten geschrieben werden.

Symbol	Bedeutung	Beispiel
=	gleich	IF A=B Wenn A gleich B ist.
>	größer als	IF A>B Wenn A größer als B ist.
<	kleiner als	IF A<B Wenn A kleiner als B ist.
>= / =>	größer oder gleich	IF A>=B / IF A=>B Wenn A größer als B oder gleich B ist.
<= / =<	kleiner oder gleich	IF A<=B / IF A=<B Wenn A kleiner als B oder gleich B ist.
<> / ><	ungleich	IF A<>B / IF A><B Wenn A nicht gleich B ist.

Bei Zeile 30 handelt es sich um einen „bedingten Sprungbefehl". Wenn die IF-Bedingung wahr ist (A=B), springt das Programm zur Zeile 50 (GOTO 50). Ist A dagegen nicht gleich B, fährt es mit der nächsten Programmzeile, also mit Zeile 40, fort, ohne den THEN-Befehl zu lesen. Durch Zeile 40 wird, wie leicht einzusehen, zum Anfang des Programms zurückgesprungen, so daß eine neue Zufallszahl gewählt wird, und wir wieder von neuem eine Zahl raten können.



Es leuchtet uns also ein, daß „Treffer!" nur dann auf dem Bildschirm abgebildet wird, wenn A gleich B ist. A war ja bekanntlich die vom Computer gewählte Zufallszahl, während wir der Variablen B durch Eingabe an der Tastatur einen Wert zuordneten. „Treffer!" zeigt somit an, daß wir richtig geraten haben.

Nach der Anzeige von „Treffer!" geht der Computer zur Zeile 60, und das Spiel beginnt von neuem. Aber auch wenn A und B nicht gleich waren, springt der Computer wieder zum Anfang des Programms, und das Spiel wird wiederholt.

WIR VEREINFACHEN DAS PROGRAMM

Programme sollten immer so einfach wie möglich geschrieben werden, damit sie leichter zu verstehen sind und die Wahrscheinlichkeit von Tippfehlern möglichst gering ist. Nun wollen wir uns unter diesem Gesichtspunkt das Rate-Programm noch einmal vornehmen.

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

Im Falle A=B soll uns der Computer „Treffer!" auf dem Bildschirm abbilden. Auf den ersten Blick sieht dies wie eine einzige Aktion aus. Wenn wir uns die Sache jedoch genauer überlegen, sehen wir, daß es eigentlich zwei Befehle sind. Der erste lautet GOTO 50 und steht in Zeile 30 nach dem Wort THEN. Der zweite Befehl befindet sich dann in Zeile 50 und lautet PRINT „Treffer!". Zur Vereinfachung wäre es jedoch besser, für eine einzige Aktion auch nur einen einzigen Befehl zu verwenden — wir sollten deshalb Zeile 30 wie folgt ändern:

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

Die Befehle von Zeile 50 und 60 brauchen wir nun nicht mehr!

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

Schon ist das Programm viel einfacher! Wenn A gleich B ist, bildet der Computer zunächst „Treffer!" auf dem Bildschirm ab, geht dann weiter zu Zeile 40 und gelangt schließlich durch den Befehl GOTO 10 wieder zum Anfang des Programms. Wenn A und B nicht gleich sind, passiert das gleiche, ohne daß jedoch „Treffer!" erscheint. Das Spiel arbeitet also auch mit nur vier Befehlen.

Nun wollen wir diese Änderungen einmal auf dem Bildschirm eingeben. Zunächst ändern wir den Befehl von Zeile 30 von GOTO 50 zu PRINT „Treffer!". Zeile 50 und 60 werden wie folgt gelöscht: Man bewegt den Cursor unter Zeile 60 und gibt 50 ein. Sobald man dann RETURN drückt, ist Zeile 50 gelöscht. Auch Zeile 60 können wir in gleicher Weise löschen.

Sind die Zeilen 50 und 60 vom Bildschirm verschwunden? Nein? — Auf dem Bildschirm sind sie zwar noch zu sehen, im Memory sind sie jedoch nicht mehr gespeichert. Mit dem LIST-Befehl können wir uns davon überzeugen.

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

Das sieht schon um einiges einfacher aus!

Und zum Schluß noch einige Verfeinerungen ...
Unser Rate-Programm haben wir nun so einfach wie möglich gemacht. Um es noch weiter zu verbessern, können wir uns noch einige Zusatzfunktionen ausdenken, durch die das Programm interessanter und die Anwendung bequemer wird. Auch bei diesen Änderungen sollten wir wieder darauf achten, alles so einfach wie möglich zu halten.

Wenn A gleich B ist, bildet der Computer „Treffer!" ab. Wenn die Werte nicht gleich sind, erhalten wir dagegen keine Mitteilung. Leute, die zum ersten Mal spielen, könnten das ganze viel einfacher verstehen, wenn wir Zeile 30 wie folgt ändern:

```
30 IF A=B THEN PRINT "Treffer!" ELSE
PRINT "Falsch!"
```

ELSE bedeutet soviel wie „sonst". Wenn wir dieses Wort hinten an den bedingten Sprungbefehl anfügen, wird der betreffende Programmschritt viel klarer.

IF **Bedingung** THEN **Befehl 1** ELSE **Befehl 2**

Da dieser neue Befehl länger als 37 Zeichen ist, „läuft" er in die nächste Zeile „über", so daß Zeile 40 vorübergehend unsichtbar wird. Aber keine Sorge — im Speicher des Computers ist noch alles da: Mit dem LIST-Befehl können wir uns davor überzeugen. Und noch etwas können wir ändern, damit unsere Freunde das Spiel leichter verstehen, wenn sie es zum ersten Mal sehen:

```
20 INPUT "Rate mal(1-5) ";B
```

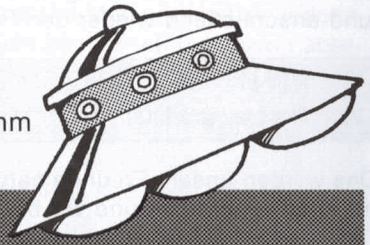

Wir werden die Bedeutung sofort verstehen, wenn wir den neuen Befehl und anschließend wieder den RUN-Befehl eingeben.

```
RUN
Rate mal(1-5) ?
```

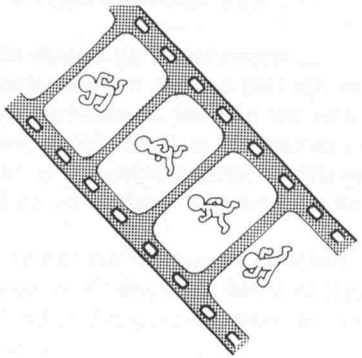
Das werden unsere Freunde natürlich viel leichter verstehen als das „?" im alten Programm, und wir brauchen gar nicht mehr viel zu erklären.

WIR BRINGEN BEWEGUNG IN DIE GRAFIK

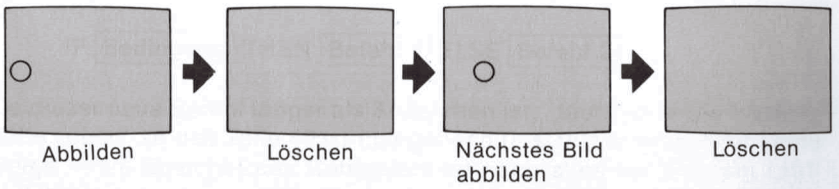
- Zeichen bewegen sich über den Bildschirm
- Ein Ufo fliegt über den Bildschirm
- Durch Variablen kann das Programm verkürzt werden



Wie jeder weiß, entsteht bei einem Film der Eindruck von Bewegung dadurch, daß viele Einzelbilder in schneller Abfolge hintereinander gezeigt werden.



Beim Fernsehen ist es genau das gleiche. Das Fernsehbild ändert sich vielmals pro Sekunde und wir haben den Eindruck, daß sich die Bilder bewegen. Da der Monitor unseres Sony Computers ähnlich wie ein Fernseher arbeitet, können wir auch auf ihm bewegende Bilder erzeugen und zwar wie folgt.



Damit sich beispielsweise das Zeichen „O" über den Bildschirm bewegt, bilden wir es zunächst ganz links ab, löschen es, bilden es etwas weiter versetzt ab, löschen es wieder usw. Bei jedem dieser Schritte müssen wir dem Computer nacheinander einen Löschbefehl, dann einen Positionierungsbefehl und schließlich noch einen Abbildungsbefehl geben.

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

Vor dem PRINT-Befehl sehen wir zwei neue Befehle. CLS bewirkt, daß alles vom Bildschirm verschwindet.

CLS + RETURN

„Was ist mit meinem Gesicht passiert?"

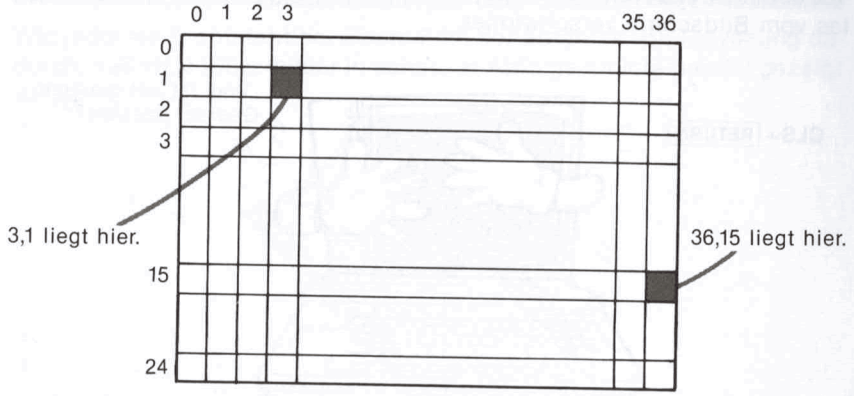
Um verstehen zu können, was der LOCATE-Befehl bewirkt, müssen wir die drei obigen Befehle zusammen eingeben.

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


                                    Z
OK
```

Durch den Befehl LOCATE 36,15 wird der Cursor um 36 Stellen nach rechts und dann um 15 Zeilen nach unten verschoben. Mit dem **Doppelpunkt** (:) teilen wir dem Computer mit, daß in der gleichen Zeile noch ein weiterer Befehl steht. Und mit dem PRINT-Befehl bewirken wir schließlich, daß an der Cursor-Position ein Z abgebildet wird.

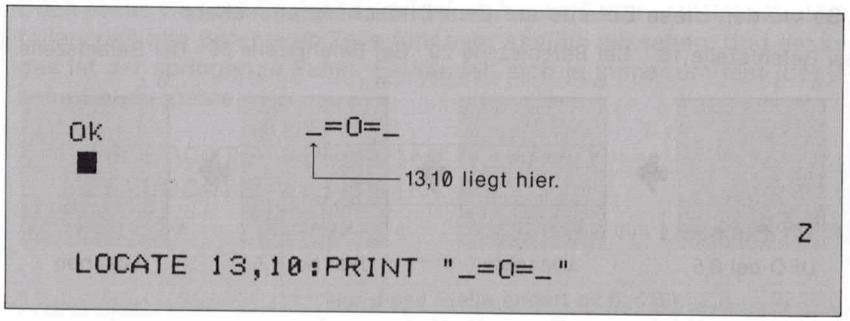
Das ganze ähnelt sehr dem PSET-Befehl, durch den uns der Computer blinkende Sterne auf dem Bildschirm abgebildet hat. Ein Unterschied besteht jedoch: Das Z ist größer als ein Punkt und außerdem handelt es sich um einen Buchstaben, während der Punkt eine Grafik ist. Für Zeichen verwenden wir folglich LOCATE 36,15: PRINT „irgendetwas", während für Grafiken der Befehl PSET (36,15) zu zu verwenden ist. Das Resultat ist in beiden Fällen gleich. 36,15 können wir als **Adresse** betrachten, an der uns der Computer etwas abbildet.



Der PRINT-Befehl, der nach dem LOCATE-Befehl und dem Doppelpunkt (:) steht, ist nicht auf ein Einzelzeichen, wie in diesem Fall „Z", beschränkt. Wir können eine **beliebige Anzahl** von Buchstaben oder auch Wörtern mit dem LOCATE-Befehl an eine bestimmte Stelle bringen. Der LOCATE-Befehl sagt dem Cursor, wohin er sich bewegen soll, bevor der PRINT-Befehl ausgeführt wird. Den LOCATE-Befehl können wir also beispielsweise verwenden, um einen Textabschnitt in der Mitte einer Seite, ein Spiel am unteren Bildschirmrand oder auch an einer beliebigen anderen Stelle beginnen zu lassen.

Doch nun zu einem etwas phantasievolleren Spiel. Gleich wird auf dem Bildschirm ein UFO erscheinen. (UFOs sind ja bekanntlich unbekannte Flugobjekte — also Raumschiffe von anderen Planeten — die sicher selbst auch einen Computer and Bord haben.) Ein UFO können wir uns ganz einfach aus fünf Zeichen zusammensetzen.

```
LOCATE 13,10:PRINT "_=0=_"
```

Das sieht doch schon ganz wie ein UFO aus! Man kann natürlich auch versuchen, mit anderen Zeichen ein noch besseres UFO zusammenzubauen. Dieses UFO hier besteht aus dem Unterstreichungszeichen (_), dem Gleichheitszeichen (=) und dem Buchstaben O.

Zum Löschen des UFOs können wir etwa folgendes eingeben:

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

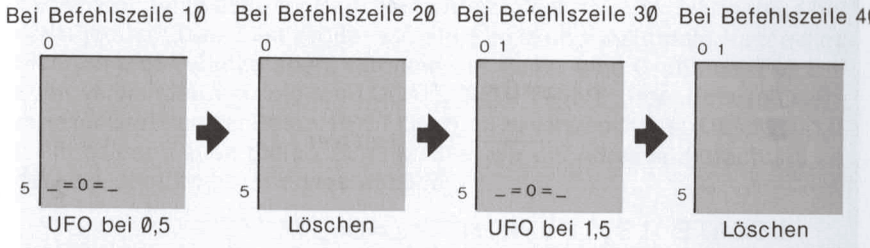
Sobald wir RETURN drücken, ist das UFO verschwunden! Die fünf Zeichen wurden durch fünf Leerstellen ersetzt. Das Löschen durch Eingabe von „Leerstellen" ist sehr nützlich; man kann Buchstaben oder ganze Wörter auf diese Weise löschen, ohne mit CLS den gesamten Bildschirm löschen zu müssen.

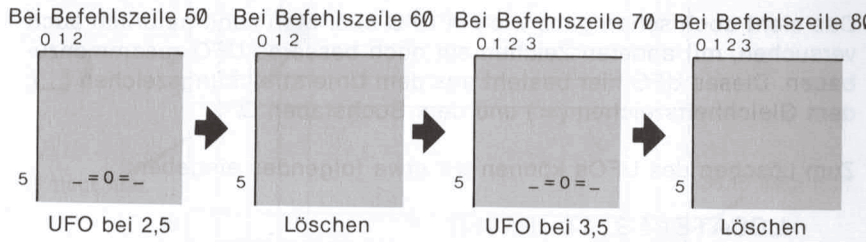
ANZEIGEN, LÖSCHEN, ANZEIGEN, LÖSCHEN ... ——

Jetzt, da wir die LOCATE- und CLS-Befehle kennen, sind wir in der Lage, bewegende Bilder zu erzeugen. Das folgende Programm läßt unser UFO über den Bildschrim fliegen. (Aber bitte noch etwas Geduld! Wir wollen uns zunächst einmal überlegen, wie das Programm funktioniert.)

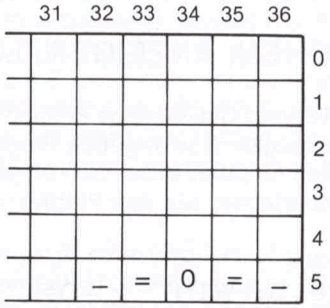
```
5 CLS
10 LOCATE 0,5:PRINT "_=0=_"
20 LOCATE 0,5:PRINT "     "
30 LOCATE 1,5:PRINT "_=0=_"
40 LOCATE 1,5:PRINT "     "
50 LOCATE 2,5:PRINT "_=0=_"
60 LOCATE 2,5:PRINT "     "
70 LOCATE 3,5:PRINT "_=0=_"
80 LOCATE 3,5:PRINT "     "
```

So würden diese Befehle auf dem Bildschirm aussehen:





Durch CLS in der ersten Programmzeile wird der Bildschirm gelöscht. Anschließend wird durch jeweils zwei Programmzeilen — 10 und 20, 30 und 40, 50 und 60, 70 und 80 — das UFO zu einer anderen Adresse verschoben. Wenn wir das ganze bis zum rechten Rand fortsetzen, ergibt sich folgendes Bild:




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

Unser Programm wäre damit ganze 67 Zeilen lang! Das Eintippen eines solchen Mammutprogramms wäre sicher keine reine Freude und viele würden es vorziehen, sich ein Fernsehprogramm anzusehen!

Doch etwas Geduld! Es geht auch viel einfacher. Es ist uns sicher aufgefallen, daß alle Befehle ab Zeile fünf sehr ähnlich aussehen. Und genau das ist der springende Punkt. Es handelt sich ja immer um fast identische Befehlspaare.

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

Dieser Teil ändert sich nie.
Immer 5.
Nur diese Stelle ändert sich: 0,0,1,1,2,2,.....32,32

Es ändert sich also nur die erste Stelle der Adresse und zwar sehr systematisch. Wie einige sicher ganz richtig vermuten, können wir hierfür eine **Variable** verwenden. Auch in diesem Beispiel zeigt sich einmal wieder, wie nützlich die Variablen sind: Das Programm verkürzt sich auf ganze sieben Zeilen!

```
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
```

Wir löschen den Speicher des Computers mit dem NEW-Befehl und geben dann dieses Programm ein. (Um später noch einige neue Zeilen einfügen zu können, lassen wir das Programm diesmal mit der Nummer 100 beginnen.) Und siehe da, das UFO fliegt!

Wenn wir uns das Programm Zeile für Zeile anschauen, verstehen wir auch, **warum** es fliegt. Zeile 100 löscht den Bildschirm für uns. (Wir wollen ja schließlich nicht, daß das UFO irgendwo anstößt.) Zeile 110 gibt die Variable an und ordnet ihr einen Wert zu. Da „I" die Horizontalposition festlegt und wir am linken Rand beginnen wollen, müssen wir „I" den **Anfangswert** 0 zuordnen. In Zeile 120 und 130 sehen wir wieder das uns schon bekannte Befehlspaar zum Anzeigen und Löschen, nun allerdings mit der Variablen „I".

In Zeile 140 wird der Variablen dann ein neuer Wert zugeordnet: I=I+1. Wie schon früher erwähnt, handelt es sich hierbei nicht um eine mathematische Gleichung (jedem echten Mathematiker würden sich die Haare sträuben), sondern um BASIC. „=" bedeutet hier also nicht „ist gleich" sondern „Zuordnung eines Wertes". „I" wird der Wert „I+1" zugeordnet, so daß sich der Wert von „I" um eins erhöht, bevor wir zur Zeile 150 kommen.

Zeile 150 bewirkt, daß der Computer zur Zeile 120 zurückspringt, wenn „I" kleiner als (<) 33 ist. Zeile 120 bildet uns wieder das UFO ab, diesmal jedoch um eine Stelle weiter nach rechts, da sich der Wert von „I" erhöht hat. Ist „I" größer als 33 geworden, so springt der Computer nicht zur Zeile 120 zurück, sondern geht zur Zeile 160 weiter. Das Programm würde eigentlich auch ohne den END-Befehl stoppen, da auf die Zeile 150 kein weiterer Befehl mehr folgt.

Auch dieses Programm enthält wieder eine **Schleife**, d.h. das Programm läuft von Zeile 120 zu 150 und dann wieder zu Zeile 120 usw. Allerdings hängt die Schleife diesmal davon ab, ob „I" kleiner als 33 ist. Eine solche Schleife nennt man **bedingte Schleife**.

Es geht sogar noch eleganter
Wir schauen uns das Programm noch einmal an.

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

Der Anfangswert ist 0.

Der Wert erhöht sich, jedesmal wenn die Schleife durchlaufen wird, und das Durchlaufen der Schleife stoppt bei 33.

Die wichtigsten Programmteile finden wir in den Zeilen 110, 140 und 150. Sie sagen dem Computer, wo er anfangen soll (bei 0) und daß er die Schleife 33 Mal durchlaufen soll. Solche Wiederholungen finden wir oft in Programmen. BASIC bietet uns hierfür eine bequeme Programmiermöglichkeiten, und zwar mit den Befehlen **FOR** und **NEXT**.

```
100 CLS
110 FOR I=0 TO 32
120 LOCATE I,5:PRINT "_=O=_"
130 LOCATE I,5:PRINT "     "
140 NEXT I
150 END
```

Wiederhole das folgende von 0 bis 32!

Erhöhe den Wert von I um 1 und springe zur Zeile 120 zurück!

Während wir zuvor drei Zeilen benötigten, kommen wir jetzt mit nur zwei Zeilen aus. die Befehle FOR in Zeile 110 und NEXT in Zeile 140 eignen sich also vorzüglich zum Programmieren von Schleifen.

Am besten merken wir uns diesen Befehl in folgender Form:

FOR **Variable** = **Anfangswert** TO **Endwert**

und

NEXT **Variable**

In unserem Programm also:

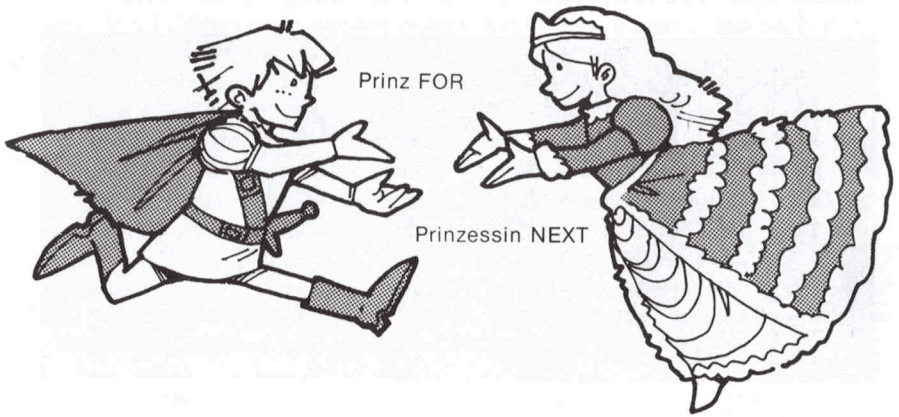
```
110 FOR I=0 TO 32
```

und

```
NEXT I
```

Der FOR-Befehl sagt dem Computer, wieviele Schleifendurchläufe er ausführen soll. Und der NEXT-Befehl zählt die Schleifendurchläufe und sorgt dafür, daß der Computer mit der nächsten Zeile weitermacht, sobald der Endwert erreicht ist.

FOR und NEXT müssen **immer** zusammen verwendet werden.



Wenn der Prinz FOR die Prinzessin NEXT nicht finden kann, schickt uns der Computer eine bedauernde Fehlermeldung und stoppt, bis wir den Fehler korrigiert haben.

WIR FÜGEN ALLES ZUSAMMEN

- Wir bringen Farbe und Ton ins Programm
- Zwei Programme werden zusammengefaßt
- Wir erstellen ein Flußdiagramm
- Wir verbessern das Programm

HINZUFÜGEN VON FARBE UND TON

Wir können die Bewegung des UFOs auch noch mit dramatischen Farben und Geräuschen untermalen. Hierzu geben wir folgendes Programm ein:

```
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
```

FOR-NEXT-Schleife

Wir können uns das Schauspiel so lange anschauen, wie wir möchten, da es sich um eine Schleifenwiederholung handelt. Doch wollen wir den Programmlauf nun mit CTRL und STOP unterbrechen, um die Befehle genauer studieren zu können.

Zeile 100 bewirkt zwei Dinge: Sie legt die Farbe (schwarzer Hintergrund, weiße Zeichen) fest und löscht den Bildschirm. In Zeile 103 wird eine neue Variable (Y) eingebracht und ihr eine **Zufallszahl** von 0 bis 32 zugeordnet. Zeile 106 ist ähnlich: Der neuen Variablen C wird ein Zufallswert zwischen 2 und 15 zugeordnet, der dann die Farbe bestimmt. (Farbe 1 können wir natürlich nicht verwenden, da schwarze Zeichen unsichtbar wären.)

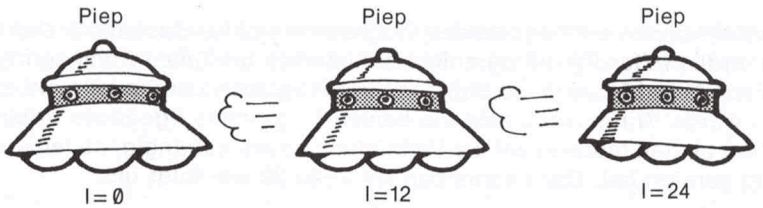
In Zeile 110 beginnt dann eine FOR-NEXT-Schleife, und jetzt ist die Variable Y ein Teil der „Adresse" des LOCATE-Befehls. Die horizontale Adresse wird durch die Variable I festgelegt, deren Wert sich **regelmäßig** von 0 bis 32 ändert, während der vertikalen Adresse durch Zeile 103 (RND) ein **Zufallswert** zugeteilt wird: Y kann einen beliebigen Wert zwischen 0 und 21 annehmen. Nun ist also auch klar, warum verschiedenfarbige UFOs in verschiedenen Höhen angeflogen kommen.

Jetzt schauen wir uns noch die Zeilen 133 und 136 an:

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

Hier kommen zwei neue Wörter vor: BEEP sagt dem Computer, daß er einen Piepton von sich geben soll — das UFO schwebt also nicht lautlos über den Bildschirm, sondern macht mit einem Piepton auf sich aufmerksam. Dieser Piepton ist aber nur zu hören, wenn A gleich 0 ist. Der Wert von A wird von I MOD 12 zugeteilt. Aber was bedeutet nun MOD? MOD ist eine Abkürzung von **Modulus** und dies wiederum bedeutet so viel wie „Rest, der beim Teilen zweier Zahlen übrigbleibt". A wird in diesem Fall der Wert zugeordnet, der beim Teilen von I durch 12 übrigbleibt. Ist I = 15, dann wird I MOD 12 zu 3. Ist I dagegen 24, so wird I MOD 12 zu 0.

Wenn sich nun I von 0 bis 32 erhöht, wird I MOD 12 bei jedem UFO dreimal zu 0, und zwar bei 0, 12 und 24. Das UFO sendet also bei seinem Flug über den Bildschirm dreimal einen Piepton aus.



RATESPIEL MIT TON UND BILD

Besonders interessant wird es, wenn wir zwei Computer-Programme zu einem Programm zusammenfassen. Ein Kombinieren von zwei oder mehr Programmen ist oft sehr nützlich. In unserem Fall lassen sich ganz verblüffende Ergebnisse erzielen.

Hier haben wir die beiden Programme noch einmal auf einen Blick:

Rate-Programm

```
10 A=INT(RND(1)*5)+1
20 INPUT "Rate mal!";B
30 IF A=B THEN PRINT "Treffer!" ELSE
PRINT "Falsch!"
40 GOTO 10
```

UFO-Programm

```
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 "_=O=_"
130 LOCATE I,Y:PRINT "      "
133 A=I MOD 12
136 IF A=0 THEN BEEP
140 NEXT I
150 GOTO 100
```

Natürlich können wir diese beiden Programme nicht einfach hintereinander anordnen. Das Rate-Programm läuft nämlich bis Zeile 40 und springt dann zur Zeile 10 zurück, so daß das UFO-Programm überhaupt nicht erreicht würde. Wir müssen also die beiden Programme irgendwie verbinden. Am besten machen wir die Verbindung davon abhängig, ob jemand richtig geraten hat. Dazu schreiben wir Zeile 30 wie folgt um:

```
30 IF A=B THEN GOTO 100 ELSE PRINT "F
alsch!"
```

Wir sind aber noch nicht ganz fertig. Wenn jemand die Zahl richtig geraten hat, fliegen die UFOs zwar los, aber sie stoppen nicht von selbst. Und wir wollen den Computer ja nicht ausschalten, sondern wieder erneut eine Zahl raten. Deshalb ändern wir Zeile 150:

```
150 GOTO 10
```


Das Programm sieht dann wie folgt aus:

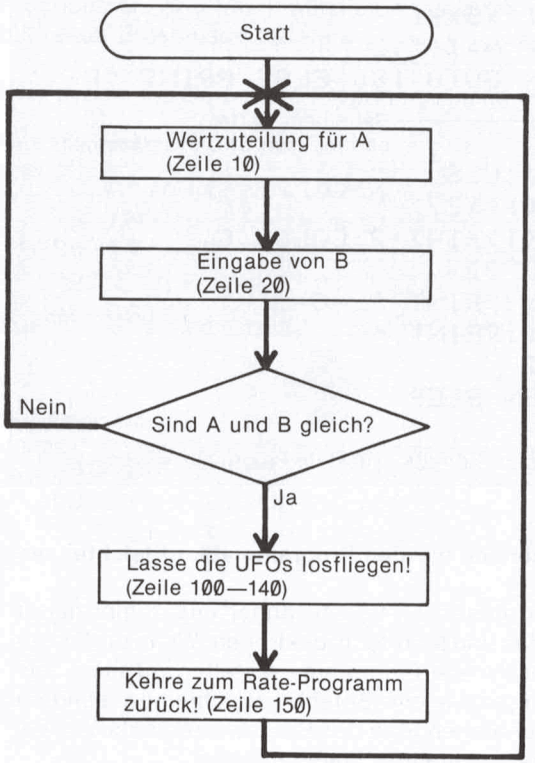
```
10 A=INT(RND(1)*5)+1
20 INPUT "Rate mal!";B
30 IF A=B THEN GOTO 100 ELSE PRINT "F
alsch!"                  <- Bei einem Treffer
40 GOTO 10                  springe zum UFO-Programmteil!
100 COLOR 15,1:CLS
103 Y=INT(RND(1)*22)
106 C=INT(RND(1)*14)+2:COLOR C
110 FOR I=0 TO 32
120 LOCATE I,Y:PRINT "_=O=_"
130 LOCATE I,Y:PRINT "      "
133 A=I MOD 12
136 IF A=0 THEN BEEP
140 NEXT I
150 GOTO 10  <- Springe zum Rate-Programmteil zurück!
```

Durch ein Flußdiagramm können wir den Programmablauf leichter verstehen ...

Unsere Programme werden allmählich immer länger und komplizierter und enthalten sogar Schleifen und Mehrfachfunktionen. Wenn professionelle Programmierer kompliziertere Programme erstellen müssen, fangen sie nicht einfach an, irgendwelche Befehle zu schreiben, sondern machen zuvor einen übersichtlichen Plan des Programmablaufs. Am besten eigent sich hierfür das sogenannte **Flußdiagramm**.

Für das Verständnis unseres Sony Computers sind aber solche Flußdiagramme nicht unbedingt erforderlich. Wer will, kann deshalb auch die folgenden Erklärungen überspringen und gleich zum nächsten Abschnitt gehen.

Damit wir wenigstens wissen, wie so ein Flußdiagramm aussieht, wollen wir hier einmal das Rate/UFO-Programm in Form eines Flußdiagramms darstellen.



Das Programm beginnt am oberen Kasten, und alle weiteren Kästen zeigen jeweils einen Vorgang an, für den ein Befehl notwendig ist (bzw. mehrere Befehle notwendig sind). Die Pfeile zwischen den Kästen deuten die Abfolge oder die Logik des Programms an. Auffallend ist weiterhin, daß aus dem parallelogrammförmigen Kasten zwei verschiedene Pfeile herauskommen; dies soll andeuten, daß es sich hier um eine Bedingung (IF) handelt, von der abhängt, an welcher Stelle das Programm fortgesetzt wird. Zu beachten ist weiterhin, daß die „Nein"-Linie sowie die unterste Linie eine Schleife einleiten, durch die das Spiel ständig wiederholt wird.

Weitere Verbesserung des Rate-Spiels

Aus einem akzeptablen Programm (ein Programm, das fehlerfrei läuft) wird erst dann ein wirklich **ausgeklügeltes** Programm, wenn man nach dem ersten erfolgreichen Lauf noch Verfeinerungen und Verbesserungen vornimmt. Auch unser Rate-Programm ist noch nicht ganz ausgereift und kann noch wie folgt verbessert werden.

- Die Nummer, die nach jedem UFO erscheint, hat die gleiche Farbe wie das UFO. Die Nummern wären aber sicher leichter zu erkennen, wenn sie alle weiß wären.
- Das Rate-Spiel beginnt immer unter dem letzten UFO — es bewegt sich also stets vertikal über den Bildschirm. Könnten wir es nicht immer an der gleichen Stelle erscheinen lassen?
- Wenn das erste UFO losfliegt, ändert sich die Hintergrundfarbe von dunkelblau zu schwarz und bleibt dann schwarz. Wäre es nicht besser, immer die gleiche Hintergrundfarbe zu haben?

Mit einigen einfachen Änderungen können alle diese Probleme gelöst werden.

```
5 COLOR 15,1                          ← Hinzufügen
7 CLS:LOCATE 0,0                      ← Hinzufügen
10 A=INT(RND(1)*5)+1
20 INPUT "Rate mal!";B
30 IF A=B THEN GOTO 100 ELSE PRINT "F
alsch!"
40 GOTO 10
100 CLS                               ← Ändern
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                            ← Ändern
```

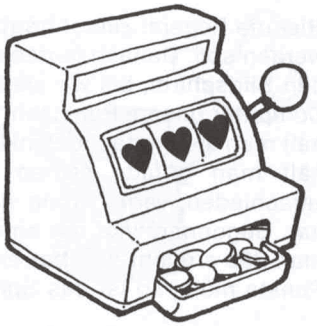
Inzwischen haben wir sicher so viel Übung im Programmieren, daß wir diese Änderungen leicht verstehen können ...

Sicher haben die meisten schon längst verstanden, was jetzt passiert. Die neuen Zeilen 5 und 7 lösen alle unsere Probleme auf elegante Weise. Zeile 5 liefert uns am Anfang weiße Zeichen auf schwarzem Hintergrund, und Zeile 7 sorgt dafür, daß das Spiel immer am oberen Bildschirmrand bleibt.

Das wichtigste aber ist, daß wir beim Ändern eines Programms folgendes immer bedenken: **Das Ändern einer bestimmten Programmstelle macht oft auch eine Änderung an einer anderen Programmstelle erforderlich.** In unserem Fall wurde der COLOR-Befehl von Zeile 100 in Zeile 5 verlegt, und in Zeile 100 darf deshalb nur noch CLS stehen bleiben. Und da das Programm bei Zeile 5 und nicht mehr bei Zeile 10 beginnt, muß auch der Sprungbefehl von Zeile 150 geändert werden.

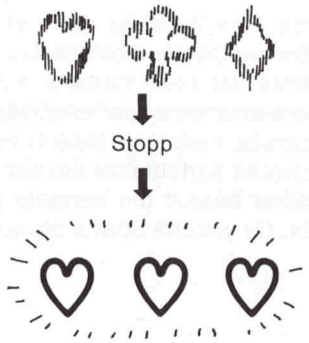
Jetzt haben wir wirklich ein ausgereiftes Programm, mit dem wir eigentlich zufrieden sein könnten. Oder könnten wir es doch vielleicht noch interessanter gestalten? ... Möglichkeiten gibt es schon noch: Beispielsweise müssen die UFOs ja nicht immer auf geraden Linien fliegen. Unserer Phantasie sind da keine Grenzen gesetzt. Bei entsprechender Programmierung werden die UFOs genauso über den Bildschirm fliegen, wie wir es uns vortsellen.

DER COMPUTER WIRD ZUM SPIELAPPARAT



- Wir programmieren einen Spielautomaten, der
  - —die angezeigten Symbole ständig ändert, bis wir ihn stoppen,
  - —die Symbole vergleicht,
  - —die Punktzahl ausrechnet.
- Verwendung von Matrixvariablen
- Verwendung des ON-GOTO-Befehls
- Verwendung von Stringvariablen
- Stoppen des Programms durch Drücken einer Taste
- Druckformat
- IF-THEN-Befehl mit mehr als einer Bedienung
- Abschnittweises Auflisten eines langen Programms

In unserem letzten Spiel ist ein UFO immer dann losgeflogen, wenn wir die richtige Zahl geraten haben. In diesem Kapitel werden wir unseren Sony Computer nun so programmieren, daß aus ihm ein Spielapparat wird, wie er auch in den Spielhöllen von Las Vegas zu finden ist! Im Gegensatz zu den Spielapparaten von Las Vegas geht es unserem Computer dabei allerdings nicht ums Geld. Alles was man gewinnen kann, ist eine hohe Punktzahl.



Hier die Regeln: Zuerst fragt der Computer, um wieviele Punkte gewettet werden soll. Danach laufen dann in schneller Abfolge die Symbole über den Bildschirm, bis wir sie stoppen. Und schließlich teilt uns dann der Computer unsere Punktzahl mit: Stimmen alle drei Symbole überein, erhält man die dreifache Punktzahl des Einsatzes. Stimmen nur zwei, so erhält man gerade seinen Einsatz zurück. Sind dagegen alle drei verschieden, verliert man doppelt soviele Punkte, wie man eingesetzt hat. Begonnen wird mit einem **Guthaben** von 100 Punkten, und sobald man 300 erreicht hat, hat man das Spiel gewonnen. Hat man gar keine Punkte mehr, so ist das Spiel verloren.

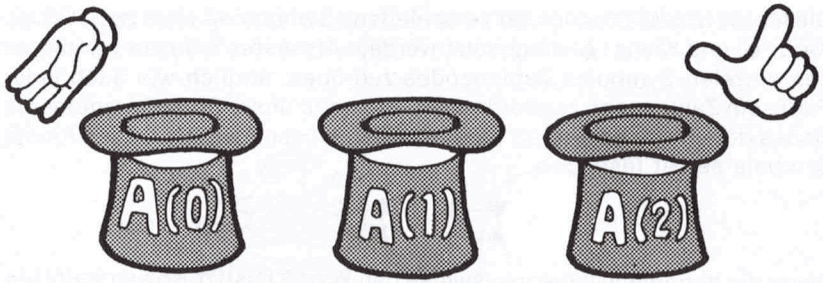
Klingt das fair? Dann wollen wir mit dem Programmieren beginnen!

Das Wichtigste in diesem Spiel sind die drei Anzeigespalten. In jeder Spalte laufen die vier Symbole (♥, ♠, ♦, ♣) so schnell vorbei, daß wir sie nicht erkennen können und die Treffer ganz vom Zufall abhängen. Selbstverständlich werden auch in diesem Fall wieder Variablen zum Ändern der Symbole benötigt.

EBENFALLS EINE GROSSE HILFE: DIE MATRIXVARIABLEN

Variablen haben wir verschiedentlich in Programmen eingesetzt, und wir erinnern uns sicher noch, daß es sich dabei um einen Buchstaben oder um einen Namen handelt, dem ein sich ändernder Wert zugeordnet wird. So wurden beispielsweise den Variablen A, B und Q Zufallszahlen zugeordnet oder auch Werte, die sich aus der Anzahl der Schleifendurchläufe ergaben.

Für unser jetziges Programm brauchen wir eine Variable, der vier verschiedene Symbole zugeordnet werden können. Da die drei Spalten den gleichen Satz von Symbolen besitzen sollen, ist eine Variable ausreichend. Allerdings sollen die einzelnen Spalten voneinander unabhängig sein: manchmal stimmen die Symbole überein, meistens jedoch nicht. Wir benötigen in diesem Fall dreimal die gleiche Variable für die drei verschiedenen Stellen. Im Unterschied zu früher besitzt die Variable jetzt aber eine Nummer in Klammern, die angibt, für welche Spalte sie vorgesehen ist.



A(Ø), A(1) und A(2) nennt man **Matrixvariable.** Auf den ersten Blick sieht das ganz schön kompliziert aus, und man fragt sich, warum man nicht gleich drei verschiedene Variablen verwendet. Bald werden wir jedoch sehen, daß diese Matrixvariablen sehr flexibel sind und das Programmieren entscheidend vereinfachen.

In unserem Fall erhalten wir ein „Feld", das drei Variablen **breit** ist, und dessen **Länge** der Anzahl der Werte entspricht, die wir zuordnen. Am besten stellen wir uns die Matrixvariablen als **mehrdimensionale** Variablen vor, dann können wir uns auch den neuen BASIC-Befehl leichter merken: **DIM**. Im folgenden Programm wird die Zeile

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

vorkommen, durch die die Matrixvariablen A(Ø), A(1) und A(2) festgelegt werden. Wenn wir Matrixvariablen verwenden, muß ein solcher DIM-Befehl immer ganz am Programmanfang stehen, damit uns der Computer genügend Speicherplatz bereithält. (Andere Beispiele für Matrixvariablen: **DIM A(1Ø)**, also die 11 Variablen von A(Ø) bis A(1Ø) und **DIM A(2), B(2)** also die 6 Variablen von A(Ø) bis A(2) und B(Ø) bis B(2).)

Und noch etwas müssen wir über Matrixvariablen wissen: Bei der Ziffer in den Klammern kann es sich auch um eine Variable handeln, also z.B. A(I), was das Programmieren in einigen Fällen entscheidend vereinfacht.

WIR PROGRAMMIEREN EINEN SPIELAUTOMATEN

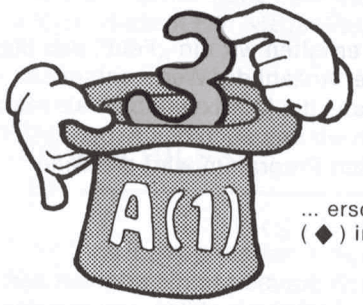
Die Matrixvariablen sollen für verschiedene Symbole — Herz (♥), Pik (♠), Karo (♦) und Kreuz (♣) eingesetzt werden. Als erstes müssen wir einmal den einzelen Symbolen Zahlencodes zuordnen, ähnlich wie auch jeder Farbe ein Zahlencode zugeordnet ist. Während unser Sony Computer die Codes der einzelnen Farben bereits kennt, müssen wir die Codes für die Symbole selbst festlegen.

♥—1 ♠—2
♦—3 ♣—4

Wenn die Variable A(1) beispielsweise den Wert 3 besitzt, so erscheint ein Kreuz in der betreffenden Spalte.

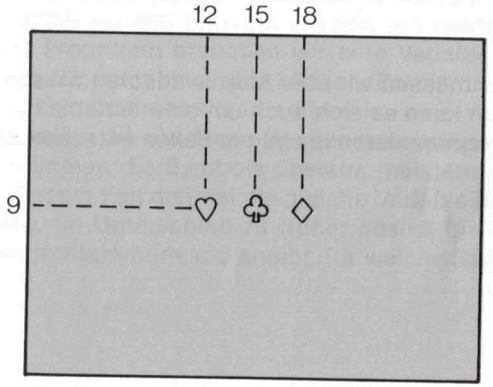
Als erstes wird ein Zahlencode vereinbart.

In diesem Fall ...



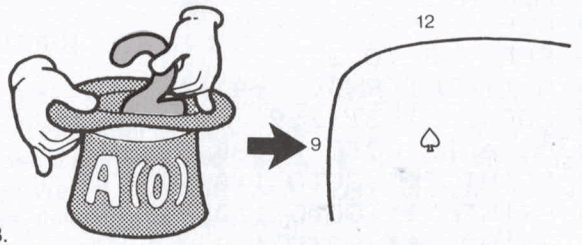
... erscheint ein Karo (♦) in der Spalte A(1)

Als nächstes müssen wir nun entscheiden, wo die Spalten auf dem Bildschirm erscheinen sollen. Am besten eigenet sich hierzu die Bildschirmmitte.

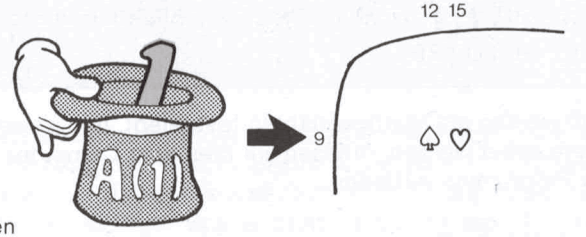


Die Spalte A(Ø) legen wir an die Stelle 12,9, die Spalte A(1) an die Stelle 15,9 und die Spalte A(2) an die Stelle 18,9.

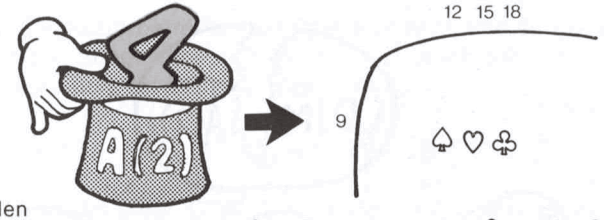
Die Planung des „Spielautomaten"-Programms ist damit abgeschlossen. Das Programm soll wie folgt ablaufen:



A(Ø) nimmt z.B. den Wert 2 an. 2 ist der Code für ♠. Somit erscheint ♠ an der Stelle 12,9.



A(1) nimmt den Wert 1 an. 1 ist der Code für ♥. Somit erscheint ♥ an der Stelle 15,9.



A(2) nimmt den Wert 4 an. 4 ist der Code für ♣. Somit erscheint ♣ an der Stelle 18,9.

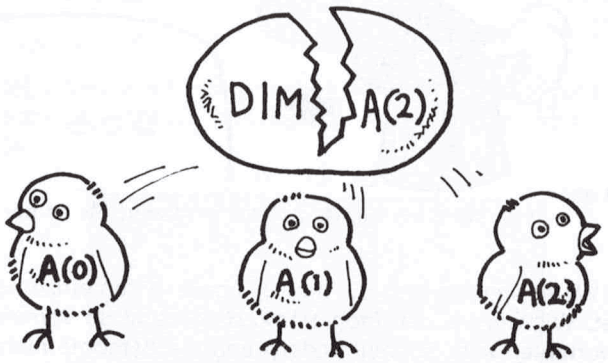
Auf diese Weise werden also die drei Symbole in den einzelnen Spalten abgebildet. Wir werden die Programmierung dann so vornehmen, daß sich die Variablen sehr schnell ändern und die Symbole nicht mehr im einzelnen erkennbar sind.

Nachdem wir die Planung des Programms nunmehr verstanden haben, können wir uns den Befehlen zuwenden.

```
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
```

In Zeile 10 werden die Matrixvariablen festgelegt. Immer wenn Matrixvariablen verwendet werden, müssen wir dies dem Computer ganz am **Anfang** des Programms mitteilen.

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



Durch Zeile 20 wird dann der Bildschirm erst mal wieder gelöscht. Auch der FOR-Befehl von Zeile 30 ist uns ja schon bekannt, und wir wissen auch, daß in diesem Fall irgendwo im Programm ein NEXT-Befehl folgen muß. Die FOR-NEXT-Schleife läuft von Zeile 30 bis Zeile 110.
In Zeile 30 kommt auch die neue Variable I (die dann später in den Klammern der Matrixvariablen verwendet wird) zum ersten Mal vor.

Auch in Zeile 40 steht im Grunde nichts Unbekanntes. Es handelt sich wieder um die Zuordnung von **Zufallszahlen**, wie sie auch bei unserem Rate-Programm vorkamen. Wenn I den **Anfangswert** 0 annimmt, sieht Zeile 40 wie folgt aus:

A(0)=INT (RND (1) * 4)+1

A(0) — der Variablen der linken Spalte — wird eine Zufallszahl zwischen 1 und 4 zugeordnet, wobei es sich bei diesen Zahlen um nichts anderes als um die Codes der Symbole ♥, ♠, ♦, ♣ handelt. Mit anderen Worten, wenn I den Wert 0 annimmt, bewirkt Zeile 40, daß irgendeines der Symbole für die linke Spalte ausgewählt wird.

Nun müssen wir dem Computer aber noch mitteilen, wo die drei Spalten auf dem Bildschirm abgebildet werden sollen.

50 LOCATE I * 3+12,9

Wenn I den Wert 0 bestizt, befindet sich der Cursor an der Stelle 12,9, und diese wird somit zur linken Spalte. Besitzt I den Wert 1, so gelangen wir zur Stelle 15,9, an der das Symbol der mittleren Spalte abgebildet wird. Und die Stelle der rechten Spalte ergibt sich für den Fall I=2 aus 2×3+12,9 zu 18,9.

Der Cursor wird also zur betreffenden Spalte bewegt, und es wurde auch schon ein Symbol ausgewählt. Um dieses nun auch wirklich auf dem Bildschirm abzubilden, ist Zeile 60 vorgesehen.

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

Hier haben wir wieder einen GOTO-Befehl, der dem Computer in diesem Fall sagt, zu den Zeilen 70, 80, 90 und 100 zu springen. Aber woher weiß der Computer nun, zu welcher dieser Zeilen er springen soll? Ganz einfach, dies wird durch den ersten Teil des Befehls — ON A(I) — entschieden. Solche ON-GOTO-Befehle werden immer wie folgt verwendet:

ON **irgendetwas** GOTO **Zeile 1**, **Zeile 2**, **Zeile 3** ...

Konkret bedeutet dies: Wenn „irgendetwas" gleich 1 ist, springe zur Zeile 1; wenn „irgendetwas" gleich 2 ist, springe zur Zeile 2; wenn „irgendetwas" gleich 3 ist, springe zur Zeile 3 ... d.h. der Wert von „irgendetwas" bestimmt, wohin gesprungen wird.

Durch ON-GOTO wird der Verkehr in verschiedene Richtungen geleitet.

Nehmen wir beispielsweise einmal an, daß A(0) den Wert 3 besitzt (Code für ♦). Zeile 60 sagt dem Computer dann, zu Zeile 90 zu springen.

```
90 PRINT "♦":GOTO 110
```

Und was passiert, wenn A(0) den Wert 2 besitzt? Dann wird durch Zeile 60 folgender Sprung bewirkt:

```
80 PRINT "♠":GOTO 110
```

Nun ist uns also klar, wie der Computer eins der vier Symbole auswählt und dann in einer der drei Spalten abbildet.

Nach jedem PRINT-Befehl springt der Computer zur Zeile 110. (In Zeile 100 ist der Befehl GOTO 110 nicht erforderlich, da der Computer ja sowieso zur nächsten Zeile weitergeht.)

In Zeile 110 wird dann der Wert von I um eins erhöht, und Zeile 120 bewirkt ein Zurückspringen zur Zeile 40. Was nun passiert, ist uns sicher klar! Wenn nicht, dann sollten wir erst einmal selbst etwas grübeln, bevor wir die anschließende Erläuterung lesen.

Alles klar? Wenn I zu 1 wird, sieht Zeile 50 wie folgt aus:

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


Dies bedeutet ja soviel wie LOCATE 15,9. Mit anderen Worten, das zufällig gewählte Symbol wird nicht mehr an der Stelle 12,9, sondern nun an der Stelle 15,9 — also in der mittleren Spalte — abgebildet. Danach wird dann der Wert von I auf 2 erhöht und wieder zur Zeile 40 zurückgesprungen. In diesem Fall erfolgt die Abbildung in der rechten Spalte, also an der Stelle 18,9. Der Computer hat das ganze Programm bereits dreimal durchlaufen und dazu nicht einmal eine Sekunde benötigt!

♡ ♤ ♢

So kann das Bild beispielsweise aussehen, aber wir können natürlich nie genau vorhersagen, welche Symbole der Computer gerade auswählt.

Die Zeilen 30 und 120 bilden eine Schleife. Der Computer durchläuft diese Schleife immer wieder von neuem, wählt mit Hilfe der Matrixvariablen Symbole aus und bildet diese dann in der linken Spalte, mittleren Spalte, rechten Spalte, linken Spalte, ... usw. ab. Da es sich um eine endlose Schleife handelt, scheinen die Symbole konstant auf dem Bildschirm „durchzulaufen", genau wie bei einem richtigen Spielautomaten.

Wenn wir es noch nicht gemacht haben, dann wollen wir nun einmal das Programm eingeben. Auch jetzt müssen wir wieder aufpassen, keine Fehler zu machen. Sobald wir dann den RUN-Befehl eingeben, setzt sich der Spielapparat in Bewegung. (Im SCREEN 0-Zustand wird die rechte Kante der Symbole nicht abgebildet.)

Aber wie können wir nun diese endlose Schleife unterbrechen, damit der Computer unsere Punktzahl mitteilen kann, und woher weiß der Computer überhaupt unsere Punktzahl? Um einen wirklich perfekten Spielapparaten zu erhalten, müssen wir das Programm noch um einige Befehle ergänzen. Hierzu benötigen wir eine ganz neue Art von Variablen — und zwar Variablen, denen kein Zahlenwert, sondern Buchstaben zugeteilt werden kann. Man nennt diese Variablen **Stringvariablen**, und mit ihnen wollen wir uns im nächsten Kapitel etwas eingehender befassen.

STRINGVARIABLEN

Wir haben bereits gelernt, daß Variablen einen beliebigen **Zahlenwert**, also beispielsweise 1, 2, 3, 191 oder 252 annehmen können. Es ist jedoch **auch** möglich, einer Variablen einen **Buchstaben** oder sogar ein aus mehreren Buchstaben bestehendes Wort zuzuteilen. (**String** bedeutet auf Deutsch „Reihe" oder „Kette" — in unserem Fall also eine Buchstabenreihe.)

An einem einfachen Beispiel, wollen wir uns einmal verdeutlichen, wie dies vor sich geht. Zunächst geben wir **PRINT A$** ein und drücken dann RETURN. ($ befindet sich über der Taste 4, so daß SHIFT gedrückt werden muß.)

```
PRINT A$

Ok
■
```

Es passiert zunächst gar nichts. Weder ein Piepton noch eine Fehlermeldung weist auf einen Fehler hin, aber der Computer bildet auch nichts ab. Dann geben wir einmal folgenden Befehl ein:

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

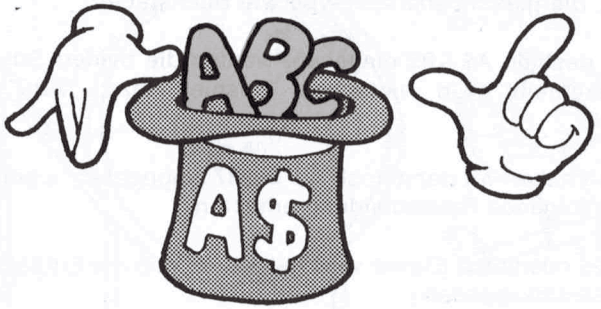
Wenn wir jetzt RETURN drücken, sollte der Bildschirm wie folgt aussehen:

```
A$="ABC"
Ok
PRINT A$
ABC
Ok
■
```

A$ ist also eine Variable mit dem Wert „ABC". Wir vergleichen dies einmal mit folgendem:

```
A=345
Ok
PRINT A
 345
Ok
■
```


Dies kennen wir ja schon. Der Variablen A wird der Zahlenwert 345 zugeordent, und in diesem Fall benötigen wir kein Dollarzeichen ($) nach dem Variablennamen A.
Ist die Regel damit klar? **Wenn der Variablenname mit einem Dollarzeichen endet, handelt es sich um einen Buchstaben oder eine Buchstabenfolge. Ohne Dollarzeichen am Ende wird der Variablen dagegen ein reiner Zahlenwert zugeordnet.**



Und noch etwas ist uns sicher aufgefallen: Bei einer Stringvariablen muß der Wert in **Anführungszeichen** stehen:

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

Anhand folgender Beispiele können wir uns noch einmal den Unterschied zwischen Zahlenvariablen und Stringvariablen verdeutlichen.

```
A=123                    A wird der Wert 123 zugeordnet.
Ok
B=234                    B wird der Wert 234 zugeordnet.
Ok
PRINT A+B                A+B wird abgebildet.
 357                     Die Summe ist 357.
Ok
A$="123"                 A$ wird der Wert „123" zugeordnet.
Ok
B$="234"                 B$ wird der Wert „234" zugeordnet.
Ok
PRINT A$+B$              A$+B$ wird abgebildet.
123234                   Die Summe ist „123234".
Ok
A$=987                   A$ wird der Wert 987 zugeordnet.
Type mismatch            Piep! (Das ist nicht erlaubt.)
Ok
■
```

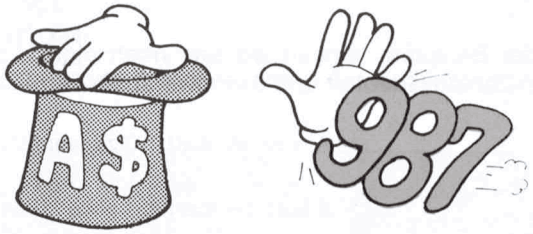
Durch A$="123" erhält die Variable A$ den Wert 123 — und dabei handelt es sich nicht um die Zahl einhundertdreiundzwanzig, sondern um die **Zeichenkette** eins-zwei-drei. In gleicher Weise wird in der nächsten Zeile der Variablen B$ der Wert 234 (Zeichenkette zwei-drei-vier) zugeordnet. Warum ist das so? Ganz einfach, wir haben ja Anführungszeichen und das Dollarzeichen verwendet. Für den Computer handelt es sich dann um eine Stringvariable, also nicht um die Zahl 123, sondern um eine Zeichenkette, die gleich behandelt wird wie Buchstaben.

Wenn wir deshalb A$ + B$ eingeben, werden die beiden Zeichenketten aneinandergereiht. Ein einfaches Beispiel dazu: "ABC" + "BCD"= "ABCBCD".

Schließlich haben wir dann noch A$ = 987 eingegeben, worauf wir vom Computer folgende Fehlermeldung erhielten:

> Type mismatch (Damit wird mitgeteilt, daß die Eingaben nicht zusammenpassen.)

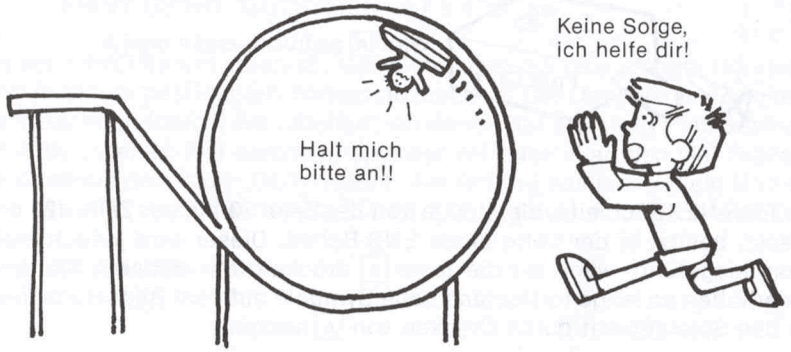
Ohne Anführungszeichen handelt es sich bei 987 um eine Zahl (neunhundertsiebenundachtzig), die **nicht** zu einer Stringvariabelen **paßt**.



Wenn also ein Variablenname mit dem Dollarzeichen endet, handelt es sich um eine Stringvariable, der ein Zeichen oder eine Zeichenkette zugeordnet wird und deren Wert in „Anführungszeichen" geschrieben wird. Doch nun zurück zu unserem Spielapparate-Programm.

UNTERBRECHEN EINER SCHLEIFE MIT DEM INKEY-BEFEHL

Bei unserem Rate-Spiel haben wir eine FOR-NEXT-Schleife verwendet, die automatisch stoppte, sobald die Variable den letzten Wert erreicht hatte. In unserem Spielapparate-Programm wird jedoch durch den GOTO 30-Befehl von Zeile 120 eine Schleife eingeleitet, die nicht stoppt.



Natürlich könnten wir den Programmlauf durch Drücken von CTRL und STOP jederzeit unterbrechen. Doch dadurch wird das ganze Programm gestoppt, und sobald wir den RUN-Befehl wieder eingeben, sind wir wieder in der Endlosschleife. Aber wir wollen ja lediglich den Spielapparat anhalten, um zu sehen, ob die Symbole übereinstimmen und wie viele Punkte wir gewonnen bzw. verloren haben. Anschließend wollen wir das Spiel dann wieder fortsetzen können.

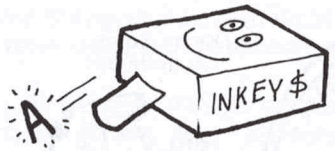
Durch Einfügen der folgenden beiden Zeilen könnte der Spielapparat auf geschickte Weise gestoppt werden:

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

Bei K$ handelt es sich natürlich wieder um eine Stringvariable. Wie wir aus Zeile 106 ersehen, bleibt der Spielapparat stehen, wenn K$ den Wert A besitzt.

Was bedeutet nun **INKEY$**? Es handelt sich um eine Funktion der Programmiersprache BASIC. INKEY bedeutet **IN**PUT vom **KEY**BOARD — d.h. es erfolgte eine Eingabe durch Drücken einer Taste an der Tastatur. Zeile 103 bewirkt also, daß **K$ ein Wert durch Drücken einer Taste an der Tastatur zugeordnet wird**. Wenn keine Taste gedrückt wird, läuft das Programm weiter, ohne daß K$ ein Wert zugeteilt wird. Wird B, C oder X gedrückt, so läuft der Spielapparat weiter. Wenn wir dagegen A (Großbuchstabe) drücken, bewirkt Zeile 106, daß der Spielaparat stoppt.

Wird keine Taste gedrückt, passiert gar nichts.



Wird [A] gedrückt, erhält man A.

Die FOR-NEXT-Schleife, die sich ja von der Zeile 30 bis zur Zeile 120 erstreckt, besitzt in der Mitte einen END-Befehl. Dieser wird jedoch erst dann ausgeführt, wenn wir die Taste [A] drücken. Mit anderen Worten: Wir erhalten so lange fortlaufend neue Symbole auf dem Bildschirm, bis wir den Spielapparat durch Drücken von [A] stoppen.

```
30 FOR I=0 TO 2
|
|
|
|
|
|
103 K$=INKEY$
106 IF K$="A" THEN END
110 NEXT I
120 GOTO 30
```

FOR-NEXT-Schleife zum Abbilden der drei Symbole in den Spalten.

Durch Drücken von [A] endet das Programm.

WIE HOCH IST UNSERE PUNKTZAHL?

Wenn der Spielapparat stoppt, möchten wir natürlich wissen, wie viele Punkte wir gewonnen (oder auch verloren) haben. Hierzu müssen wir die Regeln in das Programm einbauen. Wie waren noch die Regeln? Am Anfang haben wir ein **Guthaben** von 100 Punkten. Danach können wir dann eine beliebige Anzahl von Punkten zwischen 1 und 100 **setzen**. Die Programmzeilen hierzu sind wie folgt:

```
23 P=100:LOCATE 3,18:PRINT USING "GUT
HABEN:####";P
26 LOCATE 0,20:INPUT "WETTEINSATZ";B
```

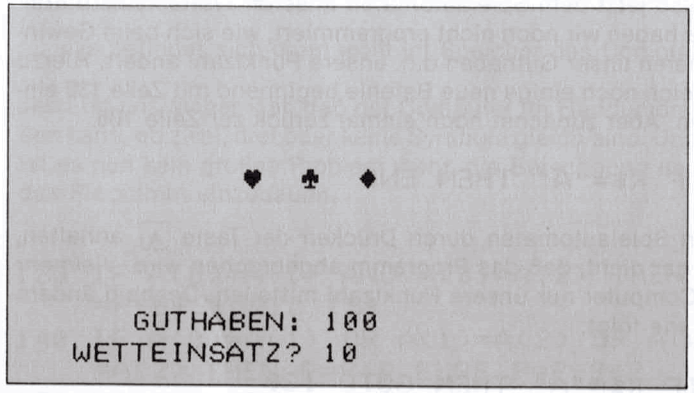

Zunächst zur Zeile 23. Da sich unser Guthaben bei jedem Spiel ändert, verwenden wir hierfür die Variable P. Dem Guthaben P wird der Anfangswert 100 zugeteilt. Der LOCATE-Befehl sorgt zusammen mit dem PRINT-Befehl dafür, daß die Punktzahl an der gewünschten Stelle auf dem Bildschirm abgebildet wird.

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

Dieser PRINT-Befehl veranlaßt den Computer, die gewünschte Information in einem bestimmten **Format** abzubilden. Die Doppelkreuz-Zeichen kennzeichnen hierbei die „Stellen", an denen der Wert von P abgebildet wird. In unserem Fall erscheint also der Wert von P an den vier Stellen direkt hinter dem Wort „GUTHABEN". Am Anfang besitzen wir ein Guthaben von 100, so daß wir an der Stelle 2,18 folgende Anzeige erhalten:

```
GUTHABEN:␣100
         ‾‾‾
         4 Stellen
```

Zeile 26 bewirkt nun, daß der Computer den von uns eingegebenen Wetteinsatz direkt unter dem Guthaben abbildet. Da sich auch der Wetteinsatz von Spiel zu Spiel ändert, verwenden wir hier ebenfalls eine Variable (B). Durch den INPUT-Befehl wird dem Computer mitgeteilt, daß er mit der Wertzuteilung für die Variable B auf eine Eingabe von unserer Tastatur warten soll. Wenn wir beispielsweise 10 setzen, sieht die Anzeige wie folgt aus:



Nach den Zeilen 23 und 26 gelangt der Computer zur FOR-NEXT-Schleife. Nun wollen wir die neuen Befehlszeilen 23, 26, 103 und 106 in unser Programm eingeben.

```
10 DIM A(2)
20 CLS
23 P=100:LOCATE 3,18:PRINT USING "GUT
HABEN:####";P
26 LOCATE 0,20:INPUT "WETTEINSATZ";B
30 FOR I=0 TO 2                    ↑ Hinzufügen
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  ← Ändern
90 PRINT "♦":GOTO 103
100 PRINT "♣"
103 K$=INKEY$
106 IF K$="A" THEN END  ← Hinzufügen
110 NEXT I
120 GOTO 30
```

Dabei müssen wir beachten, daß die GOTO 110-Befehle in den Zeilen 70, 80 und 90 zu GOTO 103 geändert werden müssen. Das wird uns auch sofort einleuchten, wenn wir bedenken, daß die neuen Programmzeilen 103, 106 eingefügt wurden.

Wir können also jetzt dem Spielautomaten unseren Wetteinsatz mitteilen, allerdings haben wir noch nicht programmiert, wie sich beim Gewinnen bzw. Verlieren unser Guthaben d.h. unsere Punktzahl ändert. Hierzu werden wir gleich noch einige neue Befehle beginnend mit Zeile 130 eingeben müssen. Aber zunächst noch einmal zurück zur Zeile 106.

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

Wenn wir den Spielautomaten durch Drücken der Taste [A] anhalten, wollen wir ja gar nicht, daß das Programm abgebrochen wird. Vielmehr soll uns der Computer nur unsere Punktzahl mitteilen. Deshalb ändern wir Zeile 106 wie folgt:

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

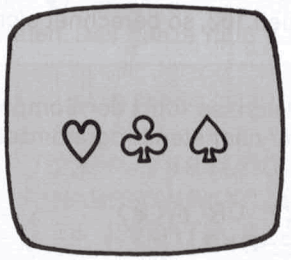
Wenn wir die Symbole auf dem Bildschirm sehen, wissen wir, ob wir gewonnen oder verloren haben. Sind alle drei gleich, gewinnen wir den dreifachen Wetteinsatz. (War das Guthaben 100 und haben wir 10 Punkte gesetzt, so erhalten wir im Falle von drei gleichen Symbolen eine neue Punktzahl von 130.) Sind zwei der Symbole gleich, gewinnen wir einmal unseren Wetteinsatz (die neue Punktzahl ist dann 110). Sind dagegen alle drei Symbole verschieden, verlieren wir das Dreifache unseres Wetteinsatzes (die neue Punktzahl ist dann 80). Erscheint z.B. ...

♥ ♣ ♠

... so wissen wir, daß wir einige Punkte verloren haben.

Natürlich betrachtet sich der Computer nicht wie wir die Symbole auf dem Bildschirm, sondern überprüft die in seinem Speicher befindlichen Variablen. Indem er die einzelnen Matrixvariablen vergleicht, weiß er sehr schnell, ob A(0), A(1) und A(2) gleich sind oder nicht.

In diesem Fall ...



... hat unser Computer folgendes gespeichert.

Und wenn auf dem Bildschirm drei Herzen zu sehen sind ...

♥ ♥ ♥

..., was befindet sich dann wohl im Speicher des Computers?

Jetzt ist uns sicher klar, daß der Computer im Handumdrehen entscheiden kann, ob zwei, drei oder keine Symbole gleich sind. Und auch für uns ist es nun kein großes Problem mehr, die Berechnung der Punktzahl in das Programm einzubauen.

```
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 12,18:PRINT USING "####";P
160 GOTO 26
```

Der neue Programmteil beginnt mit Zeile 103, d.h. die Berechnung der Punktzahl wird nur ausgeführt, wenn das Programm durch Drücken der Taste [A] gestoppt wurde.

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

Hierbei fällt auf, daß der IF-THEN-Befehl **zwei** Bedingungen besitzt, die durch **AND** — als durch und — verbunden sind. Wenn **beide** Bedingungen erfüllt sind — A(0)=A(1) und A(0)=A(2) — sind alle Matrixvariablen und somit auch die drei Symbole auf unserem Bildschirm gleich. In diesem Fall wird dann folgendes ausgeführt:

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

Der Wetteinsatz wird also mit 3 multipliziert und zum Guthaben P hinzuaddiert. Danach springt der Computer dann zur Zeile 150. Betrug beispielsweise der Wetteinsatz 10 und das Guthaben 100, so berechnet sich die Punktzahl zu 130.

Sind die drei Matrixvariablen dagegen nicht gleich, so führt der Computer den THEN-Befehl nicht aus, sondern geht zur nächsten Programmzeile weiter.

```
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
```

Hier haben wir einen IF-Befehl mit drei Bedingungen, die in diesem Fall nicht mit AND, sondern mit **OR** — als mit oder — verbunden sind. Dadurch wird folgendes bewirkt: Wenn **ein** Variablenpaar gleich ist (d.h. wenn zwei Symbole gleich sind), wird der Befehl. ...

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

ausgeführt und der Wetteinsatz zum Guthaben hinzuaddiert.
Bei einem Wetteinsatz von 10 und einem Guthaben von 100 erhält man folglich 110. Ist dagegen keine der drei Bedingungen erfüllt (d.h. alle Symbole sind verschieden), so berechnet sich die Punktzahl zu. ...

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

Wir haben also verloren! Der Wetteinsatz wird mit zwei multipliziert und dann vom Guthaben abgezogen — die neue Punktzahl wäre in diesem Fall 80.

Die Punktzahl ist nun also ausgerechnet, und es muß nur noch dafür gesorgt werden, daß sie auch angezeigt wird.

```
150 LOCATE 12,18:PRINT USING "####";P
```

Diese Zeile ist fast gleich mit dem PRINT-Formatbefehl von Zeile 23. Auf LOCATE folgt PRINT USING zur Festlegung des Formats der vier Stellen und schließlich dann der Name der Variablen, deren Wert abgebildet werden soll. Nun haben wir erreicht, was wir wollten: Jedesmal, wenn der Spielautomat gestoppt wird, berechnet der Computer die neue Punktzahl und bildet sie an der Stelle 12,18 ab.

Und was passiert danach? Das Spiel wird fortgesetzt, d.h. wir müssen einen neuen Einsatz wagen. Deshalb wird der Computer in Zeile 160 zur Zeile 26 zurückgeschickt, wo er dann darauf wartet, daß wir an der Tastatur einen neuen Wetteinsatz eingeben.

Nun wollen wir den Berechnungsteil für die Punkte in das Programm einbauen. Das ganze muß dann wie folgt aussehen:

```
10 DIM A(2)
20 CLS
23 P=100:LOCATE 3,18:PRINT USING "GUT
HABEN:####";P
26 LOCATE 0,20:INPUT "WETTEINSATZ";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   <-- Ändern
110 NEXT I
120 GOTO 30                   <-- Hinzufügen
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 12,18:PRINT USING "####";P
160 GOTO 26
```

Aber Moment noch! Das Programm ist ja zu lange für unseren Bildschirm. Wenn wir den LIST-Befehl eingeben, verschwindet der Anfang vom Bildschirm. Das Problem läßt sich jedoch leicht lösen. Wir brauchen dem Computer lediglich zu sagen, **was** er abbilden soll. z.B.:

```
LIST 10-100
```

Durch diesen Befehl werden die Zeilen 10 bis 100 abgebildet. Zur Abbildung des letzten Teils geben wir ein:

```
LIST 110-
```

Bevor wir das Programm laufen lassen, sollten wir mit diesem LIST-Befehlen noch einmal alles auf Fehler überprüfen. Wenn alles stimmt, geben wir dann den RUN-Befehl ein — und können endlich mit dem Spiel beginnen.

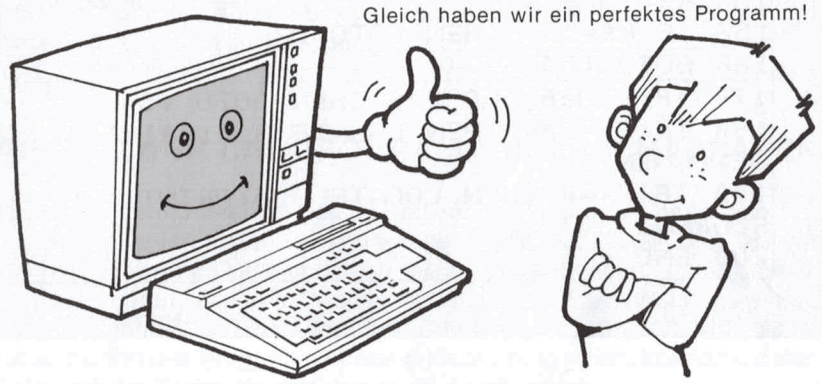
DAS PROGRAMM BEKOMMT DEN LETZTEN SCHLIFF

Aus unserem Sony Computer ist nun ein Spielautomat geworden, mit dem wir eigentlich ganz zufrieden sein könnten. Das Programm leistet eine ganze Menge: Es wird nach dem Wetteinsatz gefragt, die Symbole der drei Spalten werden geändert, und die Punktzahl wird beim Stoppen der Symbole berechnet. Es ist uns auch gelungen, das Programm recht kurz zu halten: 20 Zeilen sind nicht gerade viel, wenn man bedenkt, daß der Computer ja **jedesmal** einen Befehl erhalten muß, wenn eine Abbildung, eine Berechnung, ein Stoppen, ein Warten auf den Wetteinsatz usw. erfolgen soll.

Aber etwas fehlt noch: Wir hatten ja eigentlich geplant, daß man bei Erreichen von 300 Punkten das Spiel gewonnen hat und daß dagegen das Spiel verloren ist, wenn die Punktzahl zu 0 wird. Mit dem momentanen Programm kann es aber vorkommen, daß wir 300 Punkte überschreiten bzw. 0 Punkte unterschreiten, ohne daß das Spiel zu Ende ist.

Durch einige zusätzliche Befehle können wir das Programm noch so verbessern, daß es uns mitteilt, wenn man gewonnen bzw. verloren hat. Um das Programm wirklich **perfekt** zu machen, können wir dabei gleich noch einige andere Änderungen vornehmen:

- Das Spiel wird noch interessanter, wenn wir nach Drücken der Leertaste einen neuen Wetteinsatz eingeben können.
- Außerdem wäre es bequemer, zum Stoppen des Spiels statt der Taste A die Leertaste zu verwenden, da diese schneller zu finden ist.
- Beim Eingeben eines neuen Wetteinsatzes bleibt der alte noch auf dem Bildschirm sichtbar. Dies ist etwas verwirrend und sollte geändert werden.



Hier sind nun die erforderlichen Programmänderungen.

026 Recorder bis 031

```
10 DIM A(2)
20 CLS
23 P=100:LOCATE 3,18:PRINT USING "GUT
HABEN:####";P
24 LOCATE 11,20:PRINT"          "
26 LOCATE 0,20:INPUT "WETTEINSATZ";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
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)
```

Hinzufügen

Ändern

```
=A(2) THEN P=P+B ELSE P=P-B*2
150 LOCATE 12,18:PRINT USING "####";P
160 GOTO 26
152 GOTO 170                          <- Hinzufügen
154 K$=INKEY$
156 IF K$=" " THEN GOTO 24
160 GOTO 154                          <- Ändern
170 IF P<300 AND P>0 THEN GOTO 154
180 IF P>=300 THEN LOCATE 3,5:PRINT "
GEWONNEN!"
190 IF P<=0 THEN LOCATE 3,5:PRINT "VE
RLOREN!"
200 END
                                      <- Hinzufügen
```

Alles klar? Bevor wir die Erläuterungen lesen, sollten wir aber erst einmal einige Minuten lang versuchen, selbst dahinter zu kommen, was die neuen Programmteile bewirken.

In Zeile 170 beginnen die Befehle für „Gewinnen" und „Verlieren". Liegt die Punktzahl unter 300, **aber** über 0, so läuft das Programm weiter, d.h. das Spiel wird fortgesetzt. Liegt die Punktzahl dagegen über 300, so ist das Spiel vorbei, und oben am Bildschirm erscheint „Gewonnen!". Liegt die Punktzahl dagegen unter 0, so erscheint „Verloren!" Ist das Spiel gewonnen oder verloren, so springt der Computer zur Zeile 200, wo das Spiel durch den END-Befehl abgebrochen wird.

In Zeile 24 wird der vorhergehende Wetteinsatz vor Beginn jeder neuen Spielrunde gelöscht. Der Cursor wird zur Anzeige des Wetteinsatzes gebracht, und es werden sechs Leerstellen abgebildet.

Die Änderung in Zeile 106 ist ebenfalls sehr einfach. Der Spielautomat wird nun mit der Leertaste statt mit der Taste [A] gestoppt. Eine **Leerstelle (" ")** wird vom Computer ja bekanntlich als normales **Zeichen** behandelt.

Durch Zeile 160, 154 und 156 wird erreicht, daß das Spiel beim Drücken der Leertaste erneut startet.

Damit ist es uns also schon gelungen, die gewünschten Verbesserungen durch kleinere Änderungen und Zusätze im Programm zu erreichen.

Durch die neueingefügte Zeile 152 wird das Spiel an sich nicht geändert; diese Zeile ist nur aufgrund der anderen Programmänderungen erforderlich geworden. Der Computer wird nach der Anzeige einer neuen Punktzahl zu dem Programmteil für „Gewonnen/Verloren", das in Zeile 170 beginnt, geschickt. Wenn das Spiel weder gewonnen noch verloren ist, veranlaßt Zeile 170 einen Rücksprung zur Zeile 154, und das Spiel wird durch Drücken der Leertaste fortgesetzt.

WIR MACHEN DAS PROGRAMM ÜBERSCHAUBARER

Als wir mit dem Schreiben des Spielapparate-Programms begannen, haben wir die Zeilen in Zehnerschritten numeriert: 10, 20, 30, 40 ... Als wir dann das Programm nach und nach vervollständigt haben, füllten sich einige der „leeren" Zeilen mit neuen Befehlen. Auf diese Weise kam eine recht unregelmäßige Numerierung — etwa 90, 100, 103, 106, 110, 120 — zustande. Um das Programm übersichtlicher zu machen, können wir den Computer die Zeilen für uns **neu numerieren** lassen.

Wir geben dazu folgenden Befehl ein:

```
RENUM
```

Wir erhalten dann wieder eine Numerierung in Zehnerschritten, wobei die Reihenfolge selbstverständlich erhalten bleibt. Durch Eingabe des LIST-Befehls können wir uns das Ergebnis der neuen Numerierung anschauen. Schon ist die Numerierung viel übersichtlicher geworden. Aber was ist mit den GOTO-Befehlen geworden, in denen ja auch Zeilennummern vorkamen? Keine Sorge! Beispielsweise an Zeile 100 sehen wir, daß der Computer beim Neunumerieren an alles gedacht hat:

```
100 PRINT "♥":GOTO 140
```

Der Befehl GOTO 140 lautete vor der Neunumerierung GOTO 103. Aus Zeilennummer 103 ist nun 140 geworden und folglich muß auch GOTO 103 zu GOTO 140 geändert werden. Auch hier sehen wir wieder, wie zuverlässig unser Sony Computer arbeitet: Unser Programm läuft auch nach der Neunumerierung noch genauso gut wie vorher.

EINFÜGEN VON TITELN IN DAS PROGRAMM

Für Außenstehende—auch wenn sie sich mit BASIC schon recht gut auskennen—wird es sicher nicht einfach sein, unser Programm so ohne weiteres zu verstehen. Ja, sogar wir selbst werden Schwierigkeiten haben, alle Gedankengänge wieder zu rekonstruieren, wenn wir das Programm nach einigen Monaten zum ersten Mal wieder lesen. Dieses Problem haben nicht nur Anfänger, sondern auch erfahrene Programmierer. Aus diesem Grund bietet uns BASIC die Möglichkeit, wie wir gleich sehen werden, mit dem REM-Befehl Kommentare in das Programm einzufügen. (REM kommt von „remark" und bedeutet auf deutsch „Bemerkung"):

```
5 REM *** SPIELAPPARAT ***
7 REM
10 DIM A(2)
20 CLS
30 P=100:LOCATE 3,18:PRINT USING "GUT
HABEN:####";P
40 LOCATE 11,20:PRINT "       "
50 LOCATE 0,20:INPUT "WETTEINSATZ";B
55 '
60 ' ** STARTEN DES SPIELAPPARATS **
62 '
65 FOR I=0 TO 2          — Frühere Zeile 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 ' ** BERECHNUNG DES GUTHABENS **
182 '
185 IF A(0)=A(1) AND A(0)=A(2) THEN P
=P+B*3:GOTO 200   — Frühere Zeile 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 12,18:PRINT USING "####";P
210 GOTO 250
220 K$=INKEY$
230 IF K$=" " THEN GOTO 40
240 GOTO 220
245 '
250 ' *** GEWONNEN ODER VERLOREN ***
252 '                          Frühere Zeile 250
255 IF P<300 AND P>0 THEN GOTO 220
260 IF P>=300 THEN LOCATE 3,5:PRINT "
GEWONNEN!"
270 IF P<=0 THEN LOCATE 3,5:PRINT "VE
RLOREN!"
280 END
```

Mit REM können wir also unserem Programm einen **Titel** bzw. den Programmteilen **Untertitel** geben. Das Programm wird dadurch zwar länger, aber auch besser und schneller durchschaubar. Selbstverständlich wird das Spiel dadurch nicht im geringsten beeinflußt, schließlich handelt es sich ja nicht um Anweisungen an den Computer, sondern nur um Erläuterungen für Leute, die das Programm lesen.

Hier noch einmal einige der Kommentarzeilen:

```
5 REM *** SPIELAPPARAT ***
7 REM
```

Der REM-Befehl sagt dem Computer, daß es sich bei den folgenden Zeichen lediglich um Erläuterungen handelt, die **ignoriert** werden können. Wir können also beliebige Erläuterungen eingeben, ohne daß dadurch unser Programm geändert würde. In Zeile 5 sehen wir den Titel dieses Programms. Zeile 7 enthält nach dem REM-Befehl keinen Text. Auf diese Weise wird nach dem Titel eine Leerzeile erzeugt, die das Programm übersichtlicher machen soll. Dem gleichen Zweck dienen übrigens auch die Sterne vor und nach dem Titel: Sie sorgen dafür, daß sich der Titel deutlich von den anderen BASIC-Befehlen abhebt und sofort ins Auge springt.

Nun zu den Zeilen 55, 60 und 62. Hier sehen wir, daß sich Leerstellen auch noch einfacher erzeugen lassen: Statt des REM-Befehls wird lediglich ein ' (Apostroph) eingegeben. Diese Leerzeilen werden vom Computer einfach überlesen — weiter passiert nichts.
Die Titel, Untertitel, Sterne und Leerzeilen sind lediglich für den Benutzer und nicht für den Computer bestimmt.

Die Zeilennummer wird vom Computer allerdings in allen diesen Fällen beachtet, da sie ja vor dem ' bzw. REM-Befehl steht. So sehen wir beispielsweise in Zeile 150:

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

Der Computer springt zur Zeile 180, wo sich lediglich ein Kommentar befindet:

```
180 ' ** BERECHNUNG DES GUTHABENS **
```

Da sich in Zeile 180 kein Befehl befindet, geht der Computer weiter zur nächsten Zeile. In Zeile 182 befindet sich aber ebenfalls kein Befehl, so daß er schließlich bei Zeile 185 landet. Der **Leser** des Programms—also wir selbst oder unsere Freunde — gehen dagegen von Zeile 150 direkt zur Zeile 180 und wissen sofort, daß dies der Programmteil ist, in dem das Guthaben berechnet wird.

HERZLICHEN GLÜCKWUNSCH

Nach all den Übungen und Spielen in diesem Buch haben wir schon einiges erreicht: Die Grundlagen der neuen Sprache BASIC haben wir verstanden. Es ist ja noch gar nicht lange her, daß wir mit ganz einfachen Befehlen wie PRINT 3+5 begonnen haben, und jetzt bereiten uns selbst so komplizierte Zeilen wie A(I)=INT(RND(1) * 4)+1 oder PRINT USING "# #.# #";P kein Problem mehr. Der Computer ist uns also schon um einiges vertrauter geworden.

Aber wir wollen uns nicht auf unseren Lorbeeren ausruhen, sondern eifrig weiterüben, Programme zu schreiben. Ganz von selbst werden wir dann immer geschickter die Befehle, Grafiken, Zahlen usw. in unseren Programmen einsetzen können, und es ist nur eine Frage der Zeit, bis wir zu wirklichen **Experten** geworden sind, die überall mitreden können, wenn es um Computer geht.

Im nächsten Teil dieses Buchs werden wir noch etwas weiter mit den schon bekannten und auch mit neuen Befehlen üben. Es steht dann nichts mehr im Wege, mit neuen Farbkombinationen, Grafiken, Tönen und Spielen selbständig zu experimentieren.

Immer wenn wir **selbst Programme erstellen** wollen, sollten wir uns einige Minuten Zeit zum Planen nehmen und vielleicht ein Flußdiagramm aufstellen. Eine große Hilfe stellt dabei das Register hinten im Buch dar, in dem noch einmal die **Funktionen** und **Befehle** alphabetisch zusammengestellt sind. Das Programmieren ist im Grunde gar nicht so schwierig, wie es am Anfang schien: Unser Computer macht immer alles **genau** so, wie wir es ihm sagen und macht uns sogar auf Fehler aufmerksam. Dazu ist er auch noch ausgesprochen geduldig und vergißt alle Fehler, sobald wir den NEW-Befehl eingeben.

Oft ist es auch eine Hilfe, mit Freunden über die Programme zu sprechen. Das macht Spaß, und man erhält neue Denkanstöße zur Lösung eines Problems. Und wenn die Freunde auch einen Computer besitzen, kann man die Programme auf Band aufzeichnen oder auf ein Blatt Papier aufschreiben und mit den Freunden austauschen.

Es ist immer wieder spannend, sich etwas Neues auszudenken und dann zu sehen, ob es auch funktioniert. Die Programmiersprache BASIC führt uns in eine ganz neue Welt, in der es nicht nur auf logisches Denken, sondern auch auf Phantasie und Einfallsreichtum ankommt.

WIR ÜBEN BASIC-BEFEHLE

PRINT

```
10 PRINT "INGE"
20 PRINT "UND"
30 PRINT "PETER"
RUN
INGE
UND
PETER
```

Wir erinnern uns doch sicher noch an das folgende einfache Programm! Wir wollen es nun einmal etwas ändern:

```
10 PRINT "INGE";
20 PRINT "UND";
30 PRINT "PETER"
RUN
INGEUNDPETER
```

Hinzufügen

Mit dem ; (Semikolon) nach "PETER" und "UND" bildet uns der Computer die drei Wörter in einer Zeile ab. Wir können die drei Befehle auch in einer Programmzeile schreiben:

```
10 PRINT "INGE";"UND";"PETER"
RUN
INGEUNDPETER
```

Wichtig

Aber es ist schwierig, drei Wörter, die direkt hintereinander ohne Abstand geschrieben sind, zu lesen. Wir probieren deshalb einmal etwas anderes.

```
10 PRINT "INGE",
20 PRINT "UND",
30 PRINT "PETER
RUN
INGE          UND
PETER
```

Wichtig

14 Zeichen

Wenn wir statt des ; ein , (Komma) verwenden, folgt der erste Buchstabe des folgenden Wortes erst nach 14 Leerstellen. Das gleiche gilt auch, wenn wir die Befehle in einer Zeile eingeben.

```
10 PRINT "INGE","UND","PETER"
RUN
INGE          UND
PETER
```

Wichtig

14 Zeichen

Nun geben wir den gleichen Befehl mit einer einzigen Leerstelle zwischen den Wörtern und dem zweiten Anführungszeichen ein.

```
10 PRINT "INGE ";"UND ";"PETER"
RUN
INGE UND PETER
```

Wichtig: Auch die **Leerstelle** ist ein richtiges „Zeichen".

Zur Erinnerung schauen wir uns nun noch einmal die verschiedenen Möglichkeiten an, wie man Zahlen abbilden kann.

```
10 PRINT "3+5=";
20 PRINT 3+5
RUN
3+5= 8
```

← Abbildung von Zeichen

← Berechnung und Abbildung des Ergebnisses

Bei Zeile 10 handelt es sich um einen Befehl zur Abbildung von Zeichen und bei Zeile 20 um einen Rechenbefehl. Diese beiden Befehle werden durch ; am Ende der Zeile 10 **verbunden**. Dadurch wird dem Computer gesagt, alles in einer Zeile abzubilden, genau so, wie man die Aufgabe ja auch per Hand auf ein Blatt Papier schreiben würde. (Vor der Lösung — also vor der Zahl 8 — ist eine Leerstelle. An dieser Stelle erscheint ein Minuszeichen (−), wenn man ein **negatives Ergebnis** erhält, wie z.B. im Falle: 3−5= −2.)

Nun werden wir einmal den INPUT- und den PRINT-Befehl in dem gleichen einfachen Programm verwenden.

```
10 INPUT A
20 INPUT B
30 PRINT A,B,A+B
RUN
? 3   Wert? 3 eingeben.
? 5   Wert? 5 eingeben.
 3              5
  8
```

Wert von A — Wert von A+B — Wert von B

Der INPUT-Befehl veranlaßt den Computer, nach dem Wert zu **fragen**, der der Variablen zugeordnet werden soll, und zu warten, bis ein Wert an der Tastatur **eingegeben** wird. Nach Drücken von RETURN geht der Computer zur nächsten Zeile. Durch Zeile 30 werden die drei Werte dann schließlich auf dem Bildschirm abgebildet.

Etwas übersichtlicher könnte man das gleiche mit folgendem Programm erreichen.

```
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    ← Ergebnis der Zeile 40
```

Zeile 10 und 20 sagen dem Computer, daß A=? bzw. B=? abgebildet werden soll. Dies ist natürlich viel einfacher zu verstehen als das ? im vorigen Programm. Durch Zeile 30 wird die neue Variable C definiert, die den Wert A+B erhält. Zeile 40 bewirkt dann, daß der Computer A+B= und anschließend das Ergebnis der Rechenaufgabe abbildet.

Manchmal ist eine Abbildung übersichtlicher, wenn sich **Leerzeilen** zwischen Wörtern oder Wortgruppen befinden.


```
10 PRINT "INGE"
20 PRINT
30 PRINT "UND"
40 PRINT
50 PRINT "PETER"
RUN
INGE
                  eine Leerzeile
UND
                  eine Leerzeile
PETER
```

Zur Abbildung einer Leerzeile gibt man einfach nur den PRINT-Befehl ein, wie in den Zeilen 20 und 40 zu sehen.

INPUT

```
10 INPUT "A ist ";A
20 INPUT "B ist ";B
30 PRINT "A+B=";A+B
40 PRINT "A-B=";A-B
RUN
A ist ? 15
B ist ? 3
A+B= 18
A-B= 12
```

Hier noch einige andere Programme, in denen der INPUT- zusammen mit dem PRINT-Befehl vorkommt. In diesem Fall **fragt** also der Computer nach den Werten von A und B und **bildet** uns dann zwei Rechenaufgaben **ab**. Zusätzlich zu den beiden INPUT-Befehlen geben wir nun noch weitere Rechenaufgaben ein.

```
10 INPUT "Werte von A und B";A,B
20 PRINT "A=";A,"B=";B
30 PRINT
40 PRINT "A*B=";A*B
50 PRINT "A/B=";A/B
RUN
Werte von A und B ? 15,3
A= 15          B= 3

A*B= 45
A/B= 5
```

In Zeile 10 werden zwei verschiedenen Variablen Werte zugeordnet. Dabei ist das , (Komma) zwischen den beiden Werten (in diesem Fall 15 und 3) **sehr** wichtig. Schließlich wäre es sehr unübersichtlich, wenn der Computer statt 15,3 die Zahl 153 abbilden würde. (Welche Funktion , und ; im PRINT-Befehl der Zeile 20 haben, ist uns nun wohl auch klar. Die Zeile 20 bewirkt in diesem Beispiel die Abbildung von A=15 und B=3.)

Als nächstes werden wir nun noch eine **Stringvariable** in den INPUT-Befehl einbauen.

```
10 INPUT "Name";N$
20 PRINT N$;" ist der Beste!"
RUN
Name? Peter
Peter ist der Beste!"
```

Wenn der Variablenname mit einem $-Zeichen endet, so wird der Variablen ein Buchstabe oder ein Wort zugeordnet. Im folgenden sehen wir ein Programm, in dem sowohl eine „Buchstaben"- als auch eine „Zahlen"-Variable verwendet werden.

```
10 INPUT "Name";N$
20 INPUT "Alter";Y
30 PRINT N$;" ist ";Y;" Jahre alt."
RUN
Name? Inge
Alter? 10
Inge ist 10 Jahre alt.
```


FOR-NEXT

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

⬇

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

In diesem Programm bewirkt die FOR-NEXT-Schleife eine wiederholte Abbildung des $-Zeichens von der Position 0,10 bis zur Position 28,10. Und hier das gleiche Programm noch einmal mit einer kleinen Änderung:

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

⬇

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

Der Zusatz **STEP 3** in Zeile 20 bewirkt, daß der Cursor nach jedem abgebildeten $-Zeichen um **drei Schritte** (Step=Schritt) nach rechts bewegt wird. Mit anderen Worten, der Wert von I ändert sich nun in Dreierschritten, so daß das $-Zeichen an den Positionen 0,10 — 3,10 — 6,10 etc. und nicht wie zuvor an den Positionen 0,10 — 1,10 — 2,10 etc. erscheint. Die $-Zeichen füllen in beiden Programmen praktisch die gleiche Bildschirmfläche — nämlich von 0,10 bis **36**, 10 — aus.

Wenn der FOR-Befehl mit **STEP und einer Zahl** erweitert wird, **ändert sich die Variable in bestimmten Schritten**. Im letzten Fall wurde die Variable für eine Position verwendet, so daß sich die Position in bestimmten Schritten änderte. Im nächsten Programm werden wir dagegen mit STEP die Zählweise des Computers ändern.

```
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 läßt den Computer genau wie bei jeder anderen FOR-NEXT-Schleife zählen. Dabei ändert sich I von 50 bis auf 0 und nimmt also insgesamt 51 Werte an. Durch **STEP –5** verringert sich der Wert in Fünferschritten, so daß I insgesamt nur 11 Werte annimmt.

FOR—FOR—NEXT—NEXT: Eine Schleife in der Schleife

```
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
```

FOR ⟆ NEXT

FOR ⟆ NEXT

Mit diesem Programm soll gezeigt werden, wie in einer FOR-NEXT-Schleife noch eine Unterschleife eingebaut werden kann. Die Variable I wird durch die erste und letzte Zeile des Programms, die Variable J dagegen durch die Zeilen 30 und 50 festgelegt. Das Resultat ist, daß für jeden Wert von I viermal die Unterschleife durchlaufen wird. I ändert sich von 0 bis 2, und bei jedem I durchläuft J die Werte 0 bis 4.

Hier noch eine andere Möglichkeit der Schleifenverwendung.

```
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
```

Durch den FOR-NEXT- und STEP-Befehl ändert sich die Variable I von 0 bis N in Schritten von S. Die Werte von N und S werden hierbei mit der Tastatur eingegeben, wie wir aus dem INPUT-Befehl von Zeile 10 ersehen.

Welche Funktionen hat nun aber die Variable J und die kurze FOR-NEXT-Schleife von Zeile 60 und 70? J wird ja gar nicht ausgedruckt. Überhaupt scheint diese Schleife auf den ersten Blick keine Funktion zu haben, obwohl J insgesamt 501 Werte annimmt und die Schleife wirklich 501 mal durchlaufen wird. Wenn wir das Programm laufen lassen, wird uns klar, was diese Schleife bewirkt. Unser Sony Computer ist zwar unglaublich schnell, braucht aber dennoch eine gewisse Zeit, die 501 Schleifen zu durchlaufen, und dadurch wird die übergeordnete I-Schleife **verzögert**. Man merkt ganz deutlich, daß nach jeder Abbildung eine kurze Pause entsteht. Eine solche Schleife, die außer einer Zeitverzögerung keine weitere Funktion hat, nennt man **Blindschleife**.
Interessant wäre sicher auch, einmal eine Blindschleife zu programmieren, die den Computer noch länger beschäftigt hält. Am besten probieren wir es gleich selbst einmal aus!

IF-THEN und IF-THEN-ELSE

Hier noch ein anderes Ratespiel, wo es darum geht, eine Zahl zu raten, die der Computer vorgibt. Im Gegensatz zum Spielapparat, den wir zuvor schon einmal programmiert hatten, entscheidet hier nicht nur das reine Glück, sondern auch unsere Geschicklichkeit darüber, wie schnell wir die richtige Zahl raten. Wenn wir es nicht dem Zufall überlassen, sondern unsere Geschicklichkeit einsetzen, können wir schon nach 6 oder 7 Versuchen einen Treffer landen.

```
10 X=INT(RND(1)*100)+1
20 INPUT "Rate mal!";A
30 IF A=X THEN GOTO 70
40 IF A>X THEN PRINT "KLEINER"
50 IF A<X THEN PRINT "GRÖSSER"
60 GOTO 20
70 PRINT "Richtig!"
```

Durch die Zeilen 30, 40 und 50 wird der Computer veranlaßt, die geratene Zahl mit dem Wert von X zu vergleichen und dann eine Hilfestellung für den nächsten Versuch zu geben.

Und was hat sich nun in diesem folgenden Programm geändert?

```
10 X=INT(RND(1)*100)+1
20 INPUT "Rate mal!"
30 IF A=X THEN 70
40 IF A>X THEN PRINT "KLEINER"
50 IF A<X THEN PRINT "GRÖSSER"
60 GOTO 20
70 PRINT "Richtig!"
```

Im Grunde haben wir hier das gleiche Programm wie zuvor, auch wenn GOTO in Zeile 30 weggelassen wurde: Statt IF-THEN-GOTO können wir nämlich genausogut auch IF-THEN verwenden. Entsprechend könnten wir aber auch THEN weglassen: 30 IF A=X GOTO 70. Mit anderen Worten, um den Befehl zu verkürzen, können wir entweder THEN oder GOTO weglassen (nicht jedoch beides!).

Nun können wir noch die Zeilen 30 und 40 zu einer gemeinsamen Zeile zusammenfassen:

```
10 X=INT(RND(1)*100)+1
20 INPUT "Rate mal!";A
30 IF A=X THEN 70 ELSE
   IF A>X THEN PRINT "KLEINER"
50 IF A<X THEN PRINT "GRÖSSER"
60 GOTO 20
70 PRINT "Richtig!"
```


Aus den IF-THEN-Befehlen von Zeile 30 und 40 ist hier nun also ein einziger IF-THEN-ELSE-Befehl geworden. Obwohl Zeile 40 nun verschwunden ist, bewirkt das Programm nach wie vor das gleiche. Entsprechend müßte es nun auch möglich sein, die Zeilen 30 und 50 zu einer einzigen Zeile zusammenzulegen.

```
10 X=INT(RND(1)*100)+1
20 INPUT "Rate mal!";A
30 IF A=X THEN 70 ELSE
   IF A>X THEN PRINT "KLEINER"
   ELSE PRINT "GRÖSSER"
60 GOTO 20
70 PRINT "Richtig!"
```

Wenn wir die vier Programme laufen lassen, können wir uns selbst davon überzeugen, daß sie genau gleich sind.

DIM (Matrixvariable)

Als wir die Matrixvariablen in unserem Spielapparate-Programm verwendet haben, war folgender Befehl erforderlich ...

```
DIM A(2),
```

... in dem drei Variablen definiert wurden:

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

Danach haben wir dann für die Zahlen in den Klammern die Variable I verwendet, und diese Variable von 0 bis 2 laufenlassen (I = 0 TO 2). Der Computer durchlief eine FOR-NEXT-Schleife, und wir erhielten nach wie vor die gleichen Variablen.

Die Matrixvariablen können nicht nur ein, sondern auch mehrere Zeichen in den Klammern besitzen. Wenn wir z.B. ...

```
DIM A(3,4)
```

... eingeben, bekommen wir 20 Variablen:

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)

Ist damit klar, wie die 20 Variablen zustande kommen? Die erste Zahl in den Klammern kann vier und die zweite Zahl fünf Werte annehmen. Und 4 mal 5 ist 20! Also haben wir 20 Variablen.

Die 20 Variablen werden nicht in einer einfachen Liste, sondern in Form einer **Tabelle** dargestellt. (Die Matrixvariablen stellen wir uns am besten als zwei-dimensionale Felder vor, bei denen sich der erste Wert von links nach rechts und der zweite von oben nach unten ändert.) Wie wir uns leicht ausmalen können, eignen sich diese Matrixvariablen sehr gut zum Programmieren von Tabellen und Diagrammen. Dabei werden wir die in Klammern stehenden Zahlen der zwei-dimensionalen Matrixvariablen durch Variablen ersetzen — z.B. A(I,J). Den Nutzen von Matrixvariablen wollen wir uns einmal anhand der folgenden Tabelle verdeutlichen.

	Deutsch	Physik	Mathematik	Insgesamt
1. Klasse	68	88	70	226
2. Klasse	73	53	91	217
3. Klasse	92	98	82	272
Insgesamt	233	239	243	715
Mittelwert	77	79	81	238

Diese Tabelle zeigt die Punktzahlen eines Schülers, die er in drei Klassen erhielt. Zusammen mit der Gesamtpunktzahl und dem Mittelwert haben wir genau 20 verschiedene Zahlen. D.h. mit dem DIM (3,4)-Befehl erhalten wir genug Variablen zum Programmieren dieser Tabelle. Jetzt brauchen wir uns nur noch zu überlegen, wie wir unseren Matrixvariablen A (I,J) die Werte zuordnen können. Eine Möglichkeit wäre folgende:

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


Durch diese drei Zeilen werden die Punktzahlen im Fach Deutsch in den Computer eingegeben und den Variablen A(0,0), A(0,1) und A(0,2) zugeordnet. Jedesmal, wenn der Computer beim Durchlaufen der FOR-NEXT-Schleife an der Zeile 110 angelangt ist, wird er uns auffordern, die betreffende Punktzahl (68, 73, 92) einzugeben.

Um die Gesamtpunktzahl im Fach Deutsch auszurechnen, geben wir dann noch folgendes ein:

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

..... T ist die Variable für die Gesamtpunktzahl im Fach Deutsch.

In gleicher Weise wird die Punktzahl der ersten Klasse im Fach Physik der Variablen A(1,0) und die Punktzahl der ersten Klasse im Fach Mathematik der Variablen A(2,0) zugeordnet. Die Berechnung der in der ersten Klasse in den drei Fächern erzielte Gesamtpunktzahl und die Zuordnung zur Variablen A(3,0) erfolgt durch folgendes Programm.

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

..... T1 ist die Variable für die Gesamtpunktzahl in der ersten Klasse.

Um die anderen Punktzahlen, Gesamtpunktzahlen und Mittelwerte zu programmieren, verwenden wir ähnliche FOR-NEXT-Schleifen mit den Matrixvariablen A(I,J). Das Gesamtprogramm sieht dann wie folgt aus:

```
 10 '*PROGRAMM FÜR PUNKTZAHLENTABELLE*
 20 '
 30 DIM A(3,4)
 40 CLS
 50 '
 60 ' *** PUNKTZAHLEINGABE ***
 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 "Deutsch,Klasse"
;J+1;
110 INPUT A(0,J):GOTO 160
```

```
120 LOCATE 0,J+3:PRINT "Physik,Klasse
";J+1;
130 INPUT A(1,J):GOTO 160
140 LOCATE 0,J+6:PRINT "Mathematik,Kl
asse";J+1;
150 INPUT A(2,J)
160 NEXT J
170 NEXT I
180 '
190 ' * BERECHNUNG DER GESAMTPUNKT- *
195 ' * ZAHLEN UND MITTELWERTE      *
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 ' *** ABBILDUNG DER TABELLE ***
370 CLS
380 LOCATE 5,0
390 PRINT "DT   PH    MATHE GESAMT"
400 FOR S=1 TO 3
410 LOCATE 2,S+1:PRINT S
420 NEXT S
430 LOCATE 1,5:PRINT "GES"
440 LOCATE 1,6:PRINT "MTL"
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
```


Um unser Programm leichter erstellen zu können, haben wir vier Abschnitte geplant — Punktzahleingabe, Berechnung der Gesamtpunktzahlen/Mittelwerte und Abbildung der Tabelle. Wenn wir uns die Stellen, an denen die Variablen festgelegt werden, auf einem Blatt Papier notieren, werden wir das Programm leichter verstehen können.

Natürlich können die Matrixvariablen außer für „Spielapparate" und „Tabellen" auch noch für andere Zwecke eingesetzt werden. Und wie wir vielleicht schon vermutet haben, können auch mehr als zwei Variablen in den Klammern verwendet werden, z.B. A(P,Q,R), B(X,Y,Z,XY,XZ,YZ) oder auch jede andere Kombination aus bis zu 255 Variablen. Gerade bei Video-Spielen benutzen professionelle Programmierer oft derartige Matrixvariablen. Auch bei Buchhaltungsprogrammen von größeren Firmen oder bei anderen schwierigen Aufgaben werden diese Matrixvariablen gerne eingesetzt. Wenn wir erst noch einmal etwas mehr Übung und Programmiererfahrung haben, werden wir selbst sehen, wie nützlich die Matrixvariablen sind und sie immer wieder in unseren Programmen einsetzen. Auch beim Programmieren gilt der Grundsatz: Übung macht den Meister! Die wichtigsten Grundlagen haben wir uns nun angeeignet und können darangehen, unser Wissen allmählich zu vertiefen und zu erweitern.

Sony wünscht dabei viel Erfolg und viel Spaß!

REGISTER