Lektion 8

Das Konzept von Variablen und der Befehl `make`

Wir haben schon einiges gelernt, um kurze und gut strukturierte Programme zu entwickeln. Das Konzept der Parameter war uns dabei besonders hilfreich. Dank der Parameter können wir ein Programm schreiben, das man zum Zeichnen einer ganzen Klasse von Bildern verwenden kann. Durch die Wahl der Parameter bestimmen wir bei dem Programmaufruf einfach, welches Bild gezeichnet wird. Somit haben wir Programme zum Zeichnen von Quadraten, Rechtecken, Kreisen oder anderen Objekten mit beliebiger Größe. Es gibt aber auch Situationen, in denen uns Parameter nicht hinreichend helfen. Betrachten wir die folgende Aufgabe. Man soll eine frei wählbare Anzahl von Quadraten wie in Abb. 8.1 auf der nächsten Seite zeichnen. Wir fangen mit dem Quadrat der Größe 20×20 an und zeichnen weitere größere Quadrate, wobei das nachfolgende immer eine um zehn Schritte größere Seitenlänge haben soll als das vorherige.

Für die Zeichnung solcher Bilder können wir das Programm `QUADRAT :GR` wie folgt verwenden:

```
QUADRAT 20
QUADRAT 30
QUADRAT 40
QUADRAT 50
QUADRAT 60
...
```

Wie lang das resultierende Programm wird, hängt davon ab, wie viele Quadrate wachsender Größe man zeichnen will. Für jede gewünschte Anzahl von Quadraten muss

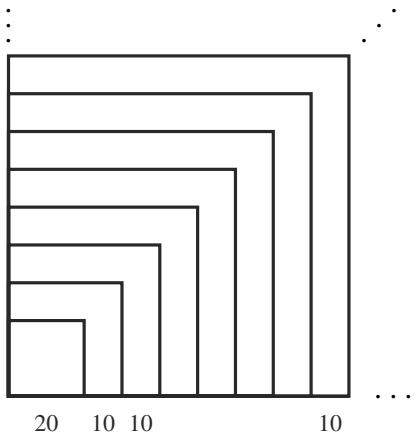


Abbildung 8.1

man ein eigenes Programm schreiben. Wir hätten aber lieber nur ein Programm anstelle von unendlich vielen, indem man die Anzahl der gezeichneten Quadrate mittels eines Parameters selbst wählen kann.

Die Idee zur Verwirklichung dieses Wunsches basiert auf dem Konzept der Variablen. In gewissem Sinne sind Variablen eine Verallgemeinerung von Parametern, wenn man dem Programm ermöglicht, während seines Laufs die Werte der Parameter zu verändern. Mit einer solchen Möglichkeit könnte man ein Programm zum Zeichnen von :AN vielen Quadraten (Abb. 8.1) wie folgt darstellen.

```
to VIELQ :GR :AN

QUADRAT :GR
Erhöhe den Wert von :GR um 10
QUADRAT :GR
Erhöhe den Wert von :GR um 10

...

QUADRAT :GR
Erhöhe den Wert von :GR um 10
end
```

} :AN - mal

Wenn wir das Programm wie oben darstellen, sehen wir sofort, dass sich die beiden Zeilen

```
QUADRAT :GR
Erhöhe den Wert von :GR um 10
```

:AN-mal wiederholen. Das ruft nach einer Schleife mit AN-vielen Durchläufen und führt somit zu folgendem Programm:

```
to VIELQ :GR :AN
repeat :AN [ QUADRAT :GR Erhöhe den Wert von :GR um 10 ]
end
```

Die Erhöhung des Wertes von :GR um 10 erreicht man durch den Befehl

```
make "GR :GR+10.
```

Wie setzt der Rechner den make-Befehl um?

Der Befehl make signalisiert ihm, dass er den Wert des Parameters ändern soll, dessen Name hinter den " nach dem Befehl make steht. Alles, was weiter rechts steht, ist ein arithmetischer Ausdruck, dessen Wert zu berechnen ist und dessen Resultat im Register mit dem Namen des zu ändernden Parameters gespeichert wird. Anschaulich kann man es folgendermaßen darstellen.

make	"A	Arithmetischer Ausdruck
Befehl zur Änderung eines Parameterwerts	Name des Parameters, dessen Wert geändert werden soll	Die Beschreibung der Rechenregel zur Bestimmung des neuen Wertes für den Parameter A.

Im Folgenden nennen wir Parameter, deren Wert sich im Laufe eines Programms ändern, nicht mehr Parameter, sondern **Variablen**. Der Name Variable signalisiert, dass es um etwas geht, was variieren kann, also um etwas Veränderliches. Der Befehl

```
make "GR :GR+10.
```

wird also von dem Rechner wie folgt umgesetzt. Der Rechner nimmt zuerst den arithmetischen Ausdruck und ersetzt :GR durch den aktuellen Wert von :GR. Dann addiert er zu diesem Wert die Zahl zehn und speichert das Resultat im Register GR. Somit liegt jetzt im Register GR eine um 10 größere Zahl als vorher.

Das Programm zum Zeichnen einer freien Anzahl von Quadraten sieht dann so aus:

```
to VIELQ :GR :AN
repeat :AN [ QUADRAT :GR make "GR :GR+10 ]
end
```

Dabei ist :GR eine Variable und :AN ein Parameter des Programms. Variable ist ein Oberbegriff. Dies bedeutet, dass die Parameter eines Programms spezielle Variablen sind, deren Werte sich während der Ausführung des Programms nicht mehr ändern.

Aufgabe 8.1 Tippe das Programm VIELQ ein und teste es für die Aufrufe VIELQ 20 20, VIELQ 100 5 und VIELQ 10 25.

Hinweis für die Lehrperson Programmieren erfordert unumgänglich das volle Verständnis des Konzepts der Variablen. Dabei ist der korrekte Umgang mit Variablen die erste ernsthafte Hürde im Programmierunterricht. Deswegen ist hier wichtig, selbständig sehr viele Aufgaben zu lösen und somit das Übungsangebot dieser Lektion zu nutzen.

Aufgabe 8.2 Mit VIELQ zeichnen wir eine Folge von Quadraten, die immer um zehn größer werden. Jetzt wollen wir die Vergrößerung 10 mittels des Parameters :ST frei wählbar machen. Kannst Du das Programm VIELQ entsprechend zu dem Programm VIELQ1 erweitern?

Aufgabe 8.3 Schreibe ein Programm zum Zeichnen einer frei wählbaren Anzahl gleichseitiger Dreiecke wie in Abb. 8.2. Dabei soll die Größe immer um den Wert fünf wachsen und das kleinste Dreieck soll zusätzlich eine frei wählbare Seitenlänge :GR haben.

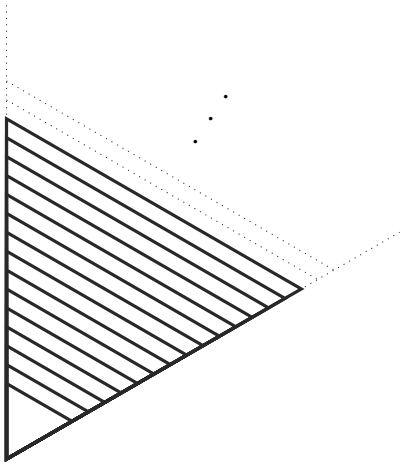
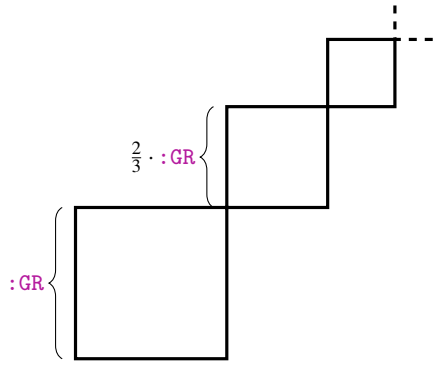


Abbildung 8.2

Beispiel 8.1 Beim Zeichnen des Bildes aus Abb. 8.1 auf Seite 138 müssten wir immer größere Quadrate aus dem selben Punkt zeichnen. Beim Zeichnen des Bildes in Abb. 8.4 auf Seite 143 ändern sich zwei Anforderungen. Erstens starten die nachfolgenden Quadrate immer in der zum Startpunkt des vorherigen Quadrats gegenüberliegenden Ecke. Zweitens wächst ihre Große nicht um einen „additiven“ Faktor +10 oder +`:ST` (Aufgabe 8.2 auf der vorherigen Seite), sondern schrumpft um den „multiplikativen“ Faktor $2/3$. Genauer bedeutet es, dass

$$\text{die Länge des nachfolgenden Quadrats} = \frac{2}{3} \cdot \text{die Länge des vorherigen Quadrats.}$$



Die Große des ersten Quadrats sowie die Anzahl der Quadrate soll frei wählbar sein. Dieses Ziel erreichen wir durch folgendes Programm:

```
to ABB8P3 :GR :AN
repeat :AN [ repeat 6 [ fd :GR rt 90 ] rt 180
             make "GR  2 * :GR / 3 ]
end
```

Wir erweitern das Programm um die Möglichkeit, den multiplikativen Faktor `:MULT` für die Verkleinerung der Seitenlänge der Quadrate frei zu wählen.

```
to ABB8P3 :GR :AN :MULT
repeat :AN [ repeat 6 [ fd :GR rt 90 ] rt 180
             make "GR  :GR * :MULT ]
end
```

Aufgabe 8.4 Teste das Programm ABB8P3, in dem du ABB8P3 folgendermaßen aufrufst: ABB8P3 100 8 0.5 und ABB8P3 20 5 1.2. Was beobachtest du?

Aufgabe 8.5 Ändere das Programm ABB8P3 so, dass die Quadratgröße um einen additiven Faktor :ADD statt um einen multiplikativen Faktor :MULT „schrumpft“.

Aufgabe 8.6 Entwerfe ein Programm zum Zeichnen der Bilder wie in Abb. 8.3. Die Anzahl der Quadrate :ANZ, die Größe des ersten Basisquadrats :GR sowie die Reduktion der Quadratgröße von Quadrat zu Quadrat :ADD soll frei wählbar sein. Danach ändere das Programm so, dass die Quadratgröße um einen multiplikativen Faktor schrumpft.

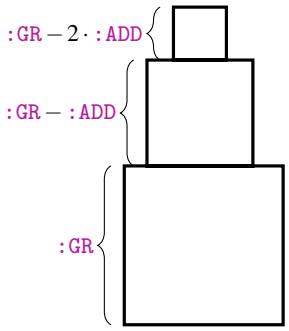


Abbildung 8.3

Aufgabe 8.7 Wir haben schon gelernt, Schnecken mit fester Größe zu zeichnen. Jetzt solltest du ein Programm schreiben, mit dem man beliebige Schnecken wie in Abb. 8.4 auf der nächsten Seite zeichnen kann.

Aufgabe 8.8 Wir haben das Programm VIELQ zum Zeichnen beliebig vieler regelmäßiger Quadrate mit einer gemeinsamen Ecke. Ähnlich sind wir in Aufgabe 8.3 vorgegangen. Anstatt regelmäßiger Quadrate haben wir regelmäßige Dreiecke mit wachsender Seitengröße gezeichnet. Entwirf jetzt ein Programm zum Zeichnen einer beliebig langen Folge von regelmäßigen Vielecken. Dabei sollen die Anfangsseitengröße, die Anzahl der Ecken und die additive Vergrößerung der Seitenlänge in jedem Schritt frei wählbar sein. Zeichne danach 20 regelmäßige 12-Ecke, deren Seitenlänge immer um fünf wächst. Welche Variablen in diesem Programm sind Parameter und welche nicht?

Bevor wir damit anfangen, die Variablen und den Befehl make intensiv zu verwenden, lernen wir zuerst, wie der Befehl make genau funktioniert.

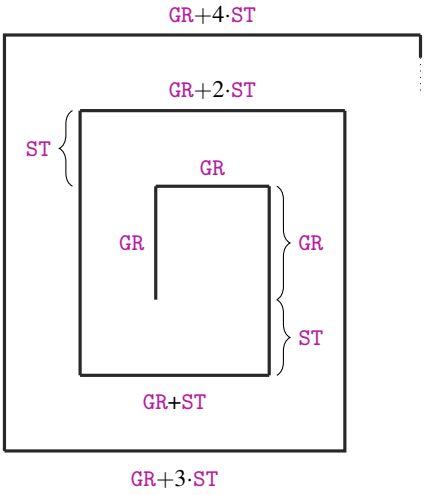


Abbildung 8.4

Wir sind in LOGO nicht gezwungen, alle im Programm verwendeten Variablen hinter `to` und dem Programmnamen aufzulisten. Wenn wir innerhalb des Programms

```
make "A  Ausdruck
```

schreiben und `:A` wurde noch nicht definiert und wurde daher auch noch nicht verwendet, benennt der Rechner ein neues Register mit `A` und ermöglicht uns damit, `:A` als Variable zu verwenden.

Wir können es mit folgendem kleinen Testprogramm `T1` überprüfen.

```
to T1
make "A 50
QUADRAT :A
end
```

Hier sehen wir, dass man das Unterprogramm `QUADRAT` mit dem Parameter `:A` im Hauptprogramm `T1` verwendet, obwohl `:A` bei der Benennung des Programms durch `to` nicht erwähnt worden ist. Aber der Befehl

```
make "A  50
```

verursacht, dass :A als Variable definiert wird und sofort den Wert 50 zugewiesen bekommt.

Aufgabe 8.9 Tippe T1 ein und teste, ob tatsächlich ein 50×50-Quadrat gezeichnet wird. Danach modifiziere T1 wie folgt:

```
to T1
make "A :A+50
QUADRAT :A
end
```

Schreibe jetzt den Befehl

```
repeat 5 [ T1 ]
```

und beobachte, was der Rechner zeichnet. Kannst du dafür eine Erklärung finden?

Schreibe jetzt folgendes Programm auf:

```
to T2 :A
make "A :A+50
QUADRAT :A
end
```

Was passiert jetzt nach dem Aufruf

```
repeat 5 [ T2 50 ]
```

Findest du eine Erklärung für den Unterschied?

Den Befehl make kann man tatsächlich dazu verwenden, um gesuchte Werte auszurechnen. Nehmen wir an, wir wollen ein Quadrat der Größe

$$X = \texttt{B} \cdot \texttt{B} - 4 \cdot \texttt{A} \cdot \texttt{C}$$

für gegebene Parameter :A, :B, :C eines Programms zeichnen. Wir könnten wie folgt vorgehen:

```
to T3 :A :B :C
make "X :B * :B
make "Y 4 * :A * :C
make "X :X - :Y
QUADRAT :X
end
```

Die Variablen :A, :B und :C des Programms T3 sind Parameter. In Tab. 8.1 verfolgen wir die Entwicklung der Speicherinhalte nach dem Bearbeiten der einzelnen Zeilen von T3 beim Aufruf T3 10 30 5. In der ersten Spalte der Tabelle sehen wir die Werte von :A, :B

	0	1	2	3	4
A	10	10	10	10	10
B	30	30	30	30	30
C	5	5	5	5	5
X	-	900	900	700	700
Y	-	-	200	200	200

Tabelle 8.1

und :C, die durch den Aufruf T3 10 30 5 eingestellt worden sind. Zu diesem Zeitpunkt gibt es noch keine Register für X und Y. Diese Tatsache notieren wir mit dem Strich -. Nach der Bearbeitung der ersten Zeile

```
make "X :B * :B
```

entsteht die Variable :X. Der Rechner setzt den Wert 30 von B in den Ausdruck :B * :B und erhält $30 \cdot 30$. Das Resultat ist 900, und dieser Wert wird im Register X abgespeichert. Bis jetzt gibt es noch kein Register mit dem Namen Y.

Bei der Bearbeitung der Programmzeile

```
make "Y 4 * :A * :C
```

wird zuerst das Register Y definiert. In den Ausdruck 4 * :A * :C setzt der Rechner die aktuellen Werte von A und C ein und erhält $4 \cdot 10 \cdot 5 = 200$. Der Wert 200 wird im Register Y abgespeichert. Bei der Bearbeitung des Programmteils

```
make "X :X - :Y
```

wird kein neues Register angelegt, weil ein Register mit dem Namen X bereits existiert. In den Ausdruck :X−:Y werden die aktuellen Werte von X und Y eingesetzt und der Rechner rechnet 900 − 200 = 700. Der Wert 700 wird im Register X abgespeichert. Durch das Speichern von 700 in X wird der alte Inhalt 900 aus dem Register X vollständig gelöscht. In der letzten Programmzeile wird nicht gerechnet, sondern nur ein Quadrat der Größe X gezeichnet. Deswegen ändern sich die Werte der Variablen durch die Ausführung dieser Zeile nicht.

Es spricht aber nichts dagegen, die Variablen X und Y sofort in der ersten Zeile des Programms wie folgt zu definieren:

```
to T3 :A :B :C :X :Y
make "X :B * :B
make "Y 4 * :A * :C
make "X :X - :Y
QUADRAT :X
end
```

Das Programm wird genau die selbe Tätigkeit ausüben. Wir müssen darauf achten, dass wir beim Aufruf von T3 fünf Zahlen angeben, z. B. T3 1 (-7) 12 0 0. Somit werden die Register X und Y von Anfang an benannt und erhalten beim Aufruf des Programms sofort den Wert 0.

Bei geschickter Klammerung kann das Programm T3 noch zum kürzeren Programm T4 abgeändert werden:

```
to T4 :A :B :C :X
make "X ( :B * :B ) - ( 4 * :A * :C )
QUADRAT :X end
```

Aufgabe 8.10 Bei der Berechnung der Seitenlänge X des Quadrates können abhängig von den eingestellten Werten von :A, :B und :C auch negative Zahlen entstehen. LOGO zeichnet in diesem Fall auch Quadrate, allerdings anders. Probiere es aus und erkläre, wie es dazu kommt.

Aufgabe 8.11 Zeichne eine Tabelle (ähnlich Tab. 8.1), in der du die Entwicklung der Speicherinhalte beim Aufruf

```
T3 20 25 10
```

dokumentierst.

Aufgabe 8.12 Betrachte das folgende Programm:

```
to TT :A :B
make "A :A+10-5
make "X :B-:A+7
make "Y 40
make "A :X-2*:Y
make "Z :B
make "X :X+:Y+:Z
make "Y :B/:X
end
```

Welche der fünf Variablen von TT sind Parameter? Zeichne wie in Tab. 8.1 die Entwicklung der Speicherinhalte jeweils nach der Ausführung einer Zeile des Programms bei folgenden Aufrufen:

a) `TT 0 10`

b) `TT 5 30`

c) `TT -5 20`

Aufgabe 8.13 Schreibe ein Programm, das zuerst ein gleichseitiges Dreieck mit der Seitenlänge 20 zeichnet. Danach zeichnet es ein regelmäßiges Viereck (Quadrat) mit der Seitenlänge 20, danach eines für ein regelmäßiges 5-Eck mit Seitenlänge 20, usw. Das nachfolgende Vieleck soll immer eine Ecke mehr haben als sein Vorgänger. Die Anzahl `:AN` der gezeichneten Vielecke soll dabei frei wählbar sein.

Beispiel 8.2 Wir sollen ein Programm `RE2ZU1 :UM` entwickeln, das Rechtecke zeichnet, deren Umfang `:UM` ist und deren horizontale Seite zweimal so lang ist wie die Vertikale.

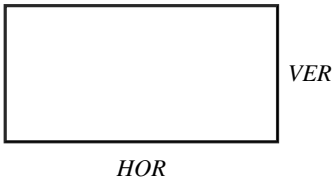


Abbildung 8.5

Wir wissen, dass (s. Abb. 8.5)

$$\texttt{UM} = 2 \cdot VER + 2 \cdot HOR \tag{8.1}$$

und

$$HOR = 2 \cdot VER \tag{8.2}$$

Wenn wir den Ausdruck $2 \cdot VER$ in der Gleichung (8.1) durch (8.2) ersetzen, erhalten wir:

$$\begin{aligned} \texttt{UM} &= 2 \cdot VER + 2 \cdot HOR \\ \texttt{UM} &= 3 \cdot HOR \\ \frac{\texttt{UM}}{3} &= HOR \end{aligned} \tag{8.3}$$

Aus (8.2) und (8.3) erhalten wir

$$VER \underset{(8.2)}{=} \frac{HOR}{2} \underset{(8.3)}{=} \frac{\texttt{UM}}{6}.$$

Mit der Formel zur Berechnung der Seitenlängen *VER* und *HOR* können wir nun das Programm schreiben:

```
to RE2ZU1 :UM
make "HOR :UM/3
make "VER :UM/6
RECHT :VER :HOR
end
```

□

Aufgabe 8.14 Die Aufgabe ist analog zu Beispiel 8.1, nur mit dem Unterschied, dass

a) die vertikalen und horizontalen Seiten gleich lang sind.

b) die horizontalen Seiten 3-mal so lang sind, wie die vertikalen Seiten.

Beim Rechnen muss man auch häufig die Quadratwurzel einer Zahl berechnen. Dazu gibt es den Befehl `sqrt` in LOGO. Mit dem Befehl

```
make "X sqrt :Y
```

wird die Wurzel des aktuellen Variablenwerts von `:Y` berechnet und im Register `:X` gespeichert. Kannst du mit Hilfe des Satzes von Pythagoras und dem Befehl `sqrt` die folgende Aufgabe lösen?

Aufgabe 8.15 Entwickle ein Programm zum Zeichnen von rechtwinkligen gleichschenkligen Dreiecken mit wählbarer Schenkellänge.

Aufgabe 8.16 Entwickle ein Programm zum Zeichnen von drei Quadraten wie in Abb. 8.6. Dabei sind nur die Seitenlängen der beiden ersten Quadrate über Parameter `:A` und `:B` gegeben. Das dritte Quadrat muss die Eigenschaft haben, dass seine Fläche der Summe der Flächen der ersten zwei kleineren Quadrate entspricht. Für das Beispiel in Abb. 8.6 stimmt es, da $5^2 = 3^2 + 4^2$ gilt.

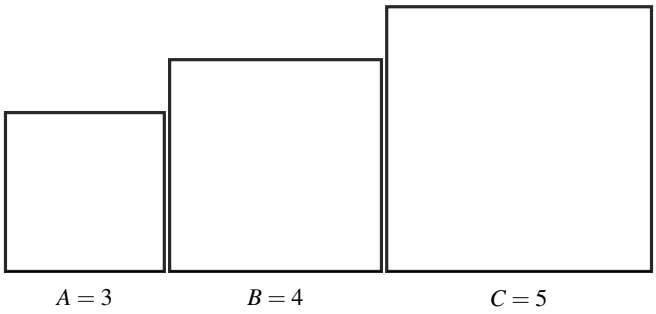


Abbildung 8.6

Aufgabe 8.17 Entwickle ein Programm mit drei Parametern `:A`, `:B` und `:C`, das eine Linie der Länge $X \cdot 10$ zeichnet, wobei X die Lösung der linearen Gleichung

$$\texttt{A} \cdot X + \texttt{B} = \texttt{C}$$

für $\texttt{A} \neq 0$ ist.

Wir können Programme mit Variablen als eine Transformation von gegebenen Eingabewerten in eine Ausgabe betrachten. Die beim Aufruf eines Programms gegebenen Variablenwerte bezeichnen wir bei dieser Sichtweise als **Eingaben** oder als **Eingabewerte** des Programms. Als die **Ausgabe** für gegebene Eingaben bezeichnen wir das Resultat der Arbeit des Programms. Somit kann die Ausgabe eines Programms ein Bild, berechnete Werte gewisser Variablen, ein Text oder auch alles zusammen sein. Zum Beispiel, beim Aufruf

```
RE2ZU1 60
```

ist `60` die Eingabe. Die Ausgabe besteht aus den Werten 20 für `:HOR`, 10 für `:VER` und dem gezeichneten Rechteck der Größe 10×20. Ob wir die Werte 10 und 20 als Ausgaben ansehen wollen, ist unsere Entscheidung.

Aufgabe 8.18 Was sind die Eingaben und Ausgaben bei dem Aufruf

```
T3 1 (-10) 5?
```

Hinweis für die Lehrperson An dieser Stelle ist es empfehlenswert den Begriff der Funktion zu thematisieren. Ein Programm berechnet eine Funktion von so vielen Argumenten, wie die Anzahl seiner Eingaben ist. Eine Funktion beschreibt wie eine Blackbox eine Beziehung zwischen Eingabewerten (Argumenten) und Ausgabewerten (Funktionswerten). Ein Programm beschreibt explizit den Rechenweg von den Eingabewerten zu den entsprechenden Ausgabewerten.

Beispiel 8.3 Die Aufgabe ist es, eine frei wählbare Anzahl :AN von Kreisen mit wachsendem Umfang zu zeichnen. Dabei soll der Umfang des kleinsten Kreises durch einen Parameter :UM frei wählbar sein und desweiteren soll die Differenz im Umfang von zwei nacheinander folgenden Kreisen durch den Parameter :NACH frei bestimmbar sein (Abb. 8.7).

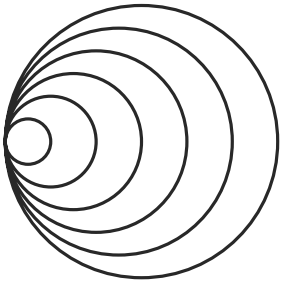


Abbildung 8.7

Wir wissen, dass unser Programm

```
to KREISE :LA
repeat 360 [ fd :LA rt 1 ]
end
```

Kreise mit dem Umfang 360·LA zeichnet. Wenn man es mit dem Wert :UM/360 aufruft, zeichnet es dann genau den Kreis mit dem Umfang :UM. Somit können wir KREISE folgendermaßen als Unterprogramm verwenden:

```
to AUGE :AN :UM :NACH
repeat :AN [ KREISE :UM/360 make "UM :UM+:NACH ]
end
```

□

Aufgabe 8.19 Welcher Wert liegt im Register `UM` nachdem das Programm `AUGE a u n` für die Zahlen `a`, `u` und `n` ausgeführt wurde? Welche Variablen in `AUGE` sind Parameter?

Aufgabe 8.20 Schreibe ein Programm, das genau zwölf Kreise mit wachsender Größe wie in Abb. 8.7 auf der vorherigen Seite zeichnet. Dabei sollen die Kreise vom kleinsten bis zu dem größten mit den Farben 1 bis 12 gezeichnet werden. Die Größe des kleinsten Kreises und der additive Größenzuwachs sollen frei wählbar sein.

Zusammenfassung

Variablen funktionieren ähnlich wie Parameter, aber zusätzlich können sie ihren Wert während der Ausführung des Programms ändern. Somit sind Parameter spezielle Variablen, die ihren Wert während des Laufs ihres Programms nicht ändern. Die Änderung des Wertes einer Variablen wird durch den Befehl `make` erreicht. Der `make`-Befehl hat zwei Argumente. Das erste Argument ist durch `"` bezeichnet und besagt, welche Variable einen neuen Wert bekommt (in welchem Register das Resultat gespeichert werden soll). Das zweite Argument ist ein arithmetischer Ausdruck, in dem Operationen über Zahlen und Variablen vorkommen dürfen. Zu den grundlegenden arithmetischen Operationen gehört neben +, −, ∗ und / auch die Quadratwurzelberechnung. Der Befehl zur Berechnung der Wurzel heißt `sqrt`. Nach `sqrt` steht ein Argument. Das Argument kann eine Zahl, eine Variable oder ein beliebiger arithmetischer Ausdruck sein.

Alle Ausdrücke wertet der Rechner so aus, dass er zuerst alle Variablennamen durch ihre aktuellen Werte ersetzt und danach das Resultat berechnet. Wenn der Rechner aus einem Register `A` den Wert für `A` ausliest, ändert sich der Inhalt in diesem Register dabei nicht. Wenn er aber in einem Register `X` die neu berechnete Zahl abspeichert, wird der alte Inhalt des Registers `X` automatisch gelöscht. Die Variablenwerte eines Programmaufrufs bezeichnen wir auch als Eingaben des Programms und das Resultat der Arbeit eines Programms bezeichnen wir als die Ausgabe des Programms.

Kontrollfragen

1. Was ist der Hauptunterschied zwischen Variablen und Parametern?

2. Erkläre, wie der Befehl `make` funktioniert.

3. Was passiert, wenn man den Befehl `make "X` ... verwendet und keine Variable mit dem Namen `:X` in dem Programm bisher definiert wurde?

4. Besteht in LOGO eine Möglichkeit, gewisse Werte aus einer vorherigen Ausführung eines Programms in die nächste Ausführung des Programms zu übertragen?

5. Wie kann man eine Wurzel in LOGO berechnen?

6. Ändert sich der Inhalt eines Registers, wenn der Rechner den Inhalt liest und zur Berechnung verwendet?

7. Was passiert mit dem ursprünglichen Inhalt eines Registers, wenn man in diesem Register einen neuen Wert speichert?

8. Lokale Parameter eines Hauptprogramms können ihre Werte während der Laufzeit des Hauptprogramms ändern. Trotzdem betrachten wir sie noch immer als Parameter. Kannst du erklären warum nicht?

Kontrollaufgaben

1. Entwickle ein Programm zum Zeichnen einer frei wählbaren Anzahl `:AN` von Treppen, wobei die nächste Treppe immer um fünf größer sein soll als die vorherige (Abb. 8.8). Die Größe der ersten Treppe `:GR` soll frei wählbar sein. Teste Dein Programm für $AN = 5$ und $GR = 20$ und dokumentiere die Entwicklung der Speicherinhalte nach der Ausführung jedes einzelnen Befehls so wie in Tab. 8.1. Welche der Variablen in Deinem Programm sind Parameter?

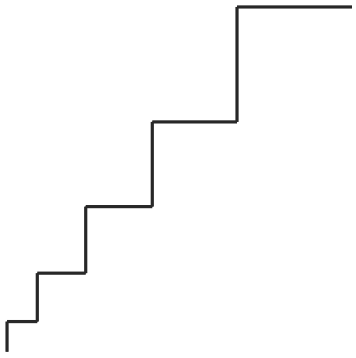


Abbildung 8.8

2. Entwickle ein Programm zum Zeichnen von Pyramiden wie in Abb. 8.9. Die Anzahl der Stufen, die Größe der Basisstufe sowie die additive Reduzierung der Stufengröße beim Übergang in eine höhere Ebene sollen frei wählbar sein.

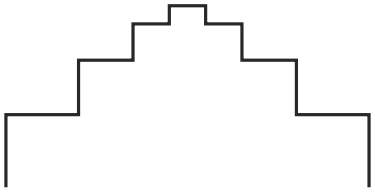


Abbildung 8.9

3. Was zeichnet das folgende Programm?

```
to SPIR :UM :ADD :AN
repeat :AN [ KREISE :UM/360 fd :UM/2 rt 20
             make "UM :UM+:ADD
             make "ADD :ADD+10 ]
end
```

   Versuche die Frage zuerst ohne einen Testlauf zu beantworten. Welche der drei Variablen in SPIR sind Parameter? Teste das Programm mit dem Aufruf SPIR 50 20 10. Dokumentiere in einer Tabelle die Inhalte der drei Register UM, ADD und AN nach jedem der zehn Durchläufe der Schleife repeat. Was steht nach der Ausführung von SPIR u a b in den Registern UM, ADD und AN?

4. Schreibe ein Programm LINGL :A :B :C :D, das ein Quadrat mit der Seitenlänge $5 \times X$ zeichnet, wobei X die Lösung der Gleichung

$$A \cdot X + B = C \cdot X + D$$

   für $A \neq C$ darstellt.

   Teste das Programm mit dem Aufruf LINGL 3 100 2 150. Welche Variablen Deines Programms sind Parameter? Dokumentiere die Änderung der Inhalte der Register nach der Ausführung der einzelnen Befehle Deines Programms während des Testlaufs LINGL 4 50 4 100.

5. Zeichne eine sechseckige Spirale wie in Abb. 8.10 auf der nächsten Seite. Die Seitenlänge wächst vom Vorgänger zum Nachfolger immer um einen festen, aber frei wählbaren Betrag :ADD. Die erste Seitenlänge ist 50. Die Anzahl der Windungen der Spirale soll frei wählbar sein.

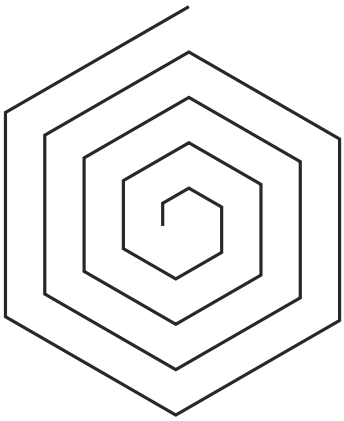


Abbildung 8.10

6. Ändere das Programm `VIEQ1 :GR :AN`, indem du den Befehl

```
make "GR :GR+10
```

durch den Befehl

```
make "GR :GR+:GR
```

austauschst. Welche Zahl liegt im Register `GR` nach der Ausführung von `VIELQ1 10 10`? Wie groß ist der Parameter `:GR` allgemein nach der Ausführung von `VIELQ1 a b`?

7. Entwickle ein Programm zum Zeichnen einer frei wählbaren Anzahl von Kreisen wie in Abb. 8.7 auf Seite 150. Dabei soll der Kreisumfang von Kreis zu Kreis immer um einen multiplikativen Faktor 1.2 wachsen. Die Größe des kleinsten Kreises soll frei wählbar sein.

8. Entwickle ein Programm zum Zeichnen einer frei wählbaren Anzahl von Halbkreisen wie in Abb. 8.11 auf der nächsten Seite. Dabei ist die Anzahl der Halbkreise mindestens 2. Der erste Halbkreis ist 100 Schritte und der zweite 120 Schritte lang. Die Länge jedes folgenden Halbkreises ist die Summe der Längen der zwei vorherigen Halbkreise.

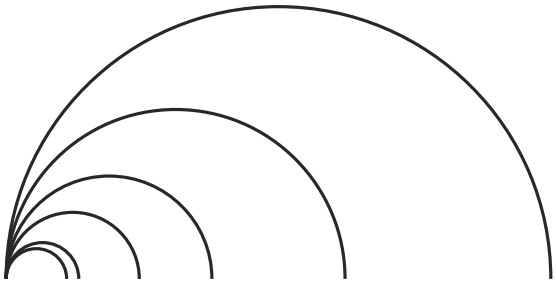


Abbildung 8.11

9. Entwickle ein Programm zum Zeichnen des Bildes aus Abb. 8.12

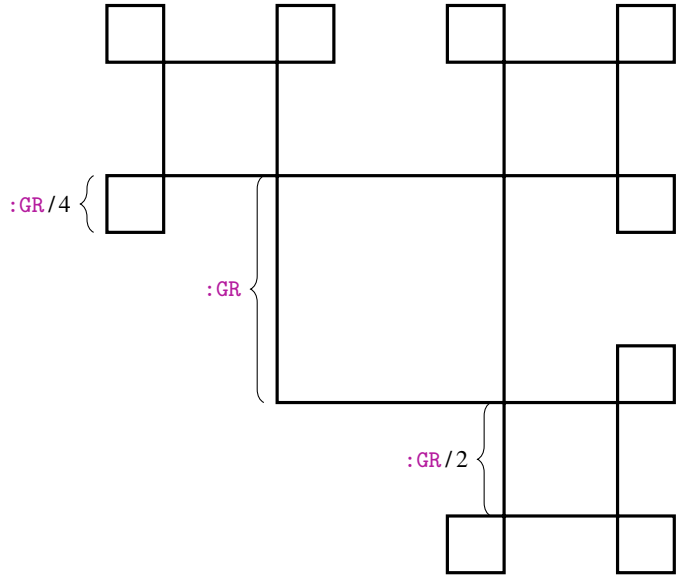


Abbildung 8.12

Lösungen zu ausgesuchten Aufgaben

Kontrollfrage 8
Die lokalen Parameter eines Hauptprogramms sind globale Parameter des Unterprogramms, indem sie definiert werden. Als solche ändern sich ihre Werte zur Laufzeit des Unterprogramms nicht. Wenn das entsprechende Unterprogramm aber mehrmals aufgerufen wird, können durch die Aufrufe die Werte der lokalen Parameter neu gesetzt werden.

Aufgabe 8.2
Wir nehmen einen neuen Parameter :ST für das Vergrößern der Seiten von Quadrat zu Quadrat. Damit können wir folgendermaßen aus VIELQ VIELQ1 machen:

```
to VIELQ1 :GR :AN :ST
repeat :AN [ QUADRAT :GR make "GR :GR+:ST ]
end
```

Wir sehen, dass es reicht, die Zahl 10 in VIELQ durch den Parameter :ST zu ersetzen.

Aufgabe 8.3
Wir bezeichnen mit :GR die Seitengröße des kleinsten gleichseitigen Dreiecks, durch :AN die Anzahl der Dreiecke und durch :ST das Vergrößern der Seitenlänge von Dreieck zu Dreieck. Dann kann unser Programm wie folgt aussehen:

```
to VIELDR :GR :AN :ST
repeat :AN [ repeat 3 [ fd :GR rt 120 ] make "GR :GR+:ST ]
end
```

Aufgabe 8.7
Das Programm zum Zeichnen von Schnecken (Abb. 8.4 auf Seite 143) kann wie folgt arbeiten:

```
to SCHNECKE :GR :AN :ST
repeat :AN [ fd :GR rt 90 fd :GR rt 90 make "GR :GR+:ST ]
end
```

Aufgabe 8.9
Das Programm T1 definiert in der to-Zeile keine Variablen. Dadurch wird der Variablen :A mit dem Aufruf des Programms T1 kein neuer Wert zugeordnet. Somit verbleibt in A der alte Wert aus dem letzten Lauf des Programms T1. Damit wird das Register A nach X Aufrufen von T1 die Zahl $X \cdot 50$ beinhalten.

Aufgabe 8.10
Sei (-100) die Zahl im Register X. Der Befehl fd :X wird jetzt als „100 Schritte zurück“ interpretiert. Er entspricht in seiner Wirkung damit dem Befehl bk 100. Auf diese Weise werden bei negativen Lösungen Quadrate der Seitenlänge $|X|$ links unten gezeichnet. Bei positiven Lösungen werden $X \times X$-Quadrate rechts oben gezeichnet.

Aufgabe 8.15
Die Grundidee ist, dass zuerst die Winkelgrößen in einem gleichschenkligen rechtwinkligen Dreieck bekannt sein müssen (Abb. 8.13 auf der nächsten Seite).

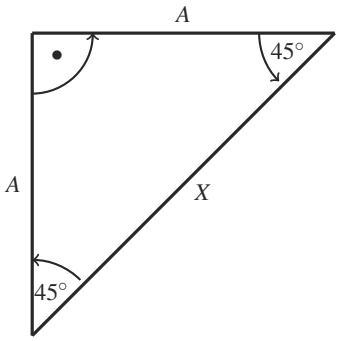


Abbildung 8.13

Der größte Winkel liegt zwischen den Schenkeln und beträgt 90°. Da in einem gleichschenkligen Dreieck zwei Winkel gleich groß sind und die Summe der Winkel in jedem Dreieck 180° beträgt, haben die beiden übrigen Winkel folglich jeweils 45°. Um das Dreieck mit Hilfe der Befehle `fd` und `rt` zu zeichnen, müssen wir noch die Länge X der Hypotenuse berechnen. Durch den Satz des Pythagoras wissen wir:

$$X^2 = A^2 + A^2$$
$$X^2 = 2 \cdot A^2$$
$$X = \sqrt{2 \cdot A^2}$$

Somit kann das Dreieck in Abb. 8.13 für ein gegebenes `:A` mit folgendem Programm `REGELSCH` gezeichnet werden:

```
to REGELSCH :A
fd :A rt 90 fd :A rt 90 rt 45
make "X sqrt ( 2*:A*:A )
fd :X
end
```

Aufgabe 8.19
In jedem Durchlauf der Schleife `repeat` des Programms `AUGE` wird der Umkreis `:UM` um `:NACH` vergrößert. Somit ist nach `:AN`-vielen Durchläufen von `repeat` der Wert von `:UM` gleich

dem ursprünglichen Wert von `UM` + `AN` · `NACH`.

Für den Aufruf `AUGE a u v` bedeutet es, dass das Register `UM` nach der Ausführung des Programms mit den Parameterwerten `a`, `u` und `v` die folgende Zahl beinhaltet:

`u` + `a` · `v`.

Kontrollaufgabe 5
Das Programm kann wie folgt arbeiten:

```
to SPIR6 :AN :ADD
make "BAS 50
repeat :AN [ fd :BAS rt 60 make "BAS :BAS+:ADD ]
end
```

Kontrollaufgabe 6
In jedem Durchlauf der Schleife `repeat` wird sich der Wert von `:GR` verdoppeln. Wenn am Anfang in `:GR` der Wert a stand, dann ist in `:GR`

nach dem ersten Schleifendurchlauf der Wert	$2 \cdot a$
nach dem zweiten Schleifendurchlauf der Wert	$4 \cdot a = 2 \cdot a + 2 \cdot a$
	$\vdots$
nach dem b-ten Schleifendurchlauf	$2^b \cdot a = 2^{b-1} \cdot a + 2^{b-1} \cdot a$

gespeichert. Falls $a = 10$ und $b = 10$ gilt, ist am Ende im Register `GR` die Zahl

$$2^{10} \cdot 10 = 1024 \cdot 10 = 10240$$

gespeichert.