Das wichtigste zu logo (ucblogo)

Daniel Mohr

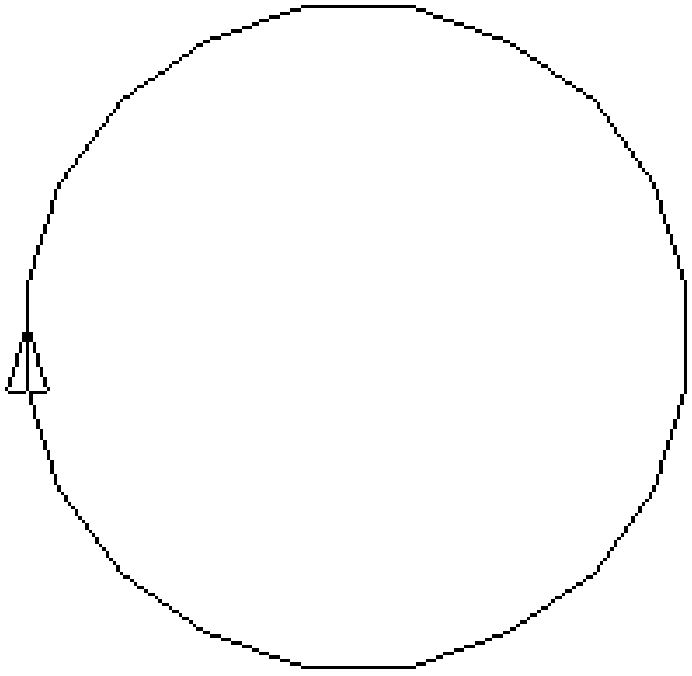
2. April 2006

Inhaltsverzeichnis

1 Ein paar Worte zur Geschichte

Die Programmiersprache *Logo* ist eine von Seymour Papert entwickelte, mit LISP verwandte Sprache aus den 60er Jahren. Als Interpretersprache ist Logo leicht zu erlernen, hat aber eine für die Zeit der Heimcomputer, als diese Sprache die größte Verbreitung fand, sehr hohe Leistungsfähigkeit, dank der dynamischen Listen aus Lisp, frei definierbaren und rekursiv aufrufbaren

Funktionen und einiger anderer Elemente. Trotzdem konnte die Programmiersprache sich nicht gegenüber anderen ersten *Anfängerprogrammiersprachen* wie z. B. *BASIC* durchsetzen, was auch daran lag, dass sie kindgerecht entwickelt und daher von vielen unterschätzt wurde.
Für die damalige Zeit sehr fortschrittliche Elemente wie z. B. eine dynamische Datentyperkennung sorgten auch für Geschwindigkeitsnachteile. Außerdem widersprach die Philosophie der Programmiersprache den damals aufkommenden Gedanken der strukturierten Programmierung: Schleifen sind normalerweise nur über Rekursion oder in Listen eingebettete Programmteile realisierbar. Rekursion galt oftmals als schwer lesbar und beansprucht sehr viel Speicher und Rechenzeit. In Daten eingebetteter und damit zur Laufzeit veränderlicher Code galt als Rezept für unvorhersagbare Programmeigenschaften und in Multiuser-Umgebungen (damit auch in Netzwerken) als Sicherheitsproblem.
In modernen Programmiersprachen finden sich die Möglichkeiten dieser Programmiersprache allerdings wieder - teilweise in Form von Objekten bzw. Klassen.
Als Designprinzip galt das Prinzip: Low floor, high ceiling, zu deutsch etwa: leichter Einstieg aber hoch hinaus.
Die Mächtigkeit und Langlebigkeit von LOGO als Lernumgebung ist Produkt seiner a) mathematischen, b) psychologisch-pädagogishen und c) softwaretechnischen Fundiertheit.



2 Turtle-Grafik

Befehl	**Kurzform**	**Beschreibung**
textscreen	`ts`	Text-Modus
fullscreen	`fs`	Graphik-Modus
splitscreen	`ss`	gemischter Modus
home		Die Schildkröte (turtle) bewegt sich zur Mitte des Bildschirms mit Ausrichtung nach oben. (0°)
clean		Der Bildschirm wird gelöscht, die Position der Schildkröte ändert sich nicht.
clearscreen	`cs`	home und clean zusammen
hideturtle	`ht`	Die Schildkröte wird unsichtbar.
showturtle	`st`	Die Schildkröte wird sichtbar.
setpencolor *farbewert*	`setpc` *farbwert*	Der Schildkröte wird die Zeichenfarbe *farbwert* zugewiesen.
penup	`pu`	Der Stift wird von der Zeichenfläche genommen.
pendown	`pd`	Der Stift wird auf die Zeichenfläche gesetzt.
right *winkel*	`rt` *winkel*	Die Schildkröte dreht sich um den Winkel *winkel* nach Rechts.
left *winkel*	`lt` *winkel*	Die Schildkröte dreht sich um den Winkel *winkel* nach Links.
forward *länge*	`fd` *länge*	Die Schildkröte bewegt sich um *länge* Schritte nach vorne.
back *länge*	`bk` *länge*	Die Schildkröte bewegt sich um *länge* Schritte zurück.

3 Dateien laden/speichern

Befehl	**Beschreibung**
`load "`*dateinameundpfad*	Die Datei *dateinameundpfad* wird geladen. Es werden alle Befehle in dieser Datei abgearbeitet.
`save "`*dateinameundpfad*	Prozeduren, Variablen etc. werden in die Datei *dateinameundpfad* gespeichert. Diese Datei kann mit `load` wieder geladen werden und der jetzige Zustand wiederhergestellt werden.

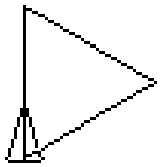
4 Prozeduren

Mit dem Schlüsselwort `to` beginnt eine Prozedur und mit dem Schlüsselwort `end` endet diese Prozedur.
Beispiel:

```
to dreieck :s
  fd :s
  rt 120
  fd :s
  rt 120
  fd :s
  rt 120
end
```

Hierbei wurde s als interne Variable in der Prozedur definiert und muß beim Aufruf übergeben werden.
Beispiel:

```
dreieck 50
```



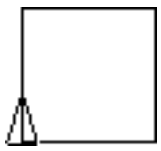
Diese Prozedur kann auch direkt in eine einfache Text-Datei geschrieben werden und anschließend mit `load` geladen werden.

Aufgabe 4.1 *Schreibe eine Prozedur* `quadrat`*, die ein Quadrat zeichnet. Dieser Prozedur wird die Seitenlänge übergeben.*

5 Wiederholungsanweisung

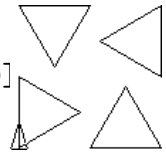
Mit `repeat` kann ein Anweisungsblock wiederholt werden.
Beispiel:

```
repeat 4 [fd 50 rt 90]
```



Dabei muß der Anweisungsblock in Eckigenklammern stehen.
Beispiel:

```
repeat 4 [dreieck 50 pu fd 100 rt 90 pd dreieck 50]
```



Aufgabe 5.1 *Schreibe eine Prozedur* `rechteck`*, die ein Rechteck zeichnet. Dieser Prozedur wird Breite und Länge übergeben.*

6 Variable, Ausgabe und Rechnen

Mit dem Schlüsselwort `make` wird eine Variable erzeugt und mit einem Wert versehen. Beispiel:

```
? make "a 5
? print :a
5
? make "b :a*2
? print :b
10
```

Nach dem Schlüsselwort `make` kommt nach einem Leerzeichen ein Anführungsstrich um einen Text (String) als Variablennamen einzuleiten. Wieder mit einem Leerzeichen getrennt folgt der Wert.
Das Schlüsselwort `print` erzeugt eine Ausgabe auf dem Bildschirm – also hier vom Wert der Variablen; auf den Wert einer Variable wird zugegriffen indem man einen Doppelpunkt vor den Variablennamen stellt.
Wenn mit einer `print`-Anweisung ein Wort ausgegeben werden soll, so muß dieses Wort mit Anrführungsstriche eingeleitet werden:

```
? print "Hallo
Hallo
```

Soll eine `print`-Anweisung mehrere Objekte ausgeben, so muß man die gesamte Anweisung in Klammern setzen und jedes Objekt muß ggf. mit Anführungsstrichen versehen werden:

```
? (print 3 "ist "eine "Zahl)
3 ist eine Zahl
? (print 3 "plus 4 "ist 7)
3 plus 4 ist 7
```

Aufgabe 6.1 *Schreibe eine Prozedur* `doppeltertext`*, die einen Text doppelt aneinander schreibt.*
Aufruf: `doppeltertext "bla`
Ausgabe: `bla bla`

Mit den im Rechner üblichen Operatoren `+ - * /` kann gerechnet werden:

```
? print 3*4                    ? print 3/4
12                             0.75
? print 3+4                    ? print 3-4
7                              -1
```

Mit der vorgegebenen Prozedur `int` bekommt man den Ganzzahlanteil einer Zahl:

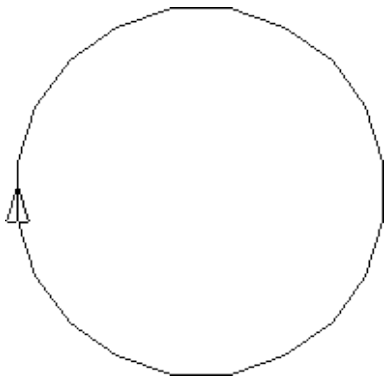
```
? print int 1.5                ? print int 3/4
1                              0
? print int -1.5               ? print int 4/3
-1                             1
```

Mit der vorgegebenen Prozedur `sqrt` kann man die Quadratwurzel einer Zahl berechnen lassen:

```
? print sqrt 4
2
? print sqrt 5
2.23606797749979
```

Beispiel einer Prozedur, die ein n-Eck zeichnet und dazu den nötigen Winkel berechnet:

```
to neck :n :s
  make "w 360/:n
  repeat :n [fd :s rt :w]
end
```



Aufruf:

```
neck 20 30
```

Beispiel:

```
to berechnequadrat :x
  output :x * :x
end
```

Das Schlüsselwort `output` gibt den berechneten Wert zurück und kann an der Stelle des Aufrufes verwendet werden:

```
? print berechnequadrat 11
121
? print (berechnequadrat 11)-20
101
```

Aufgabe 6.2 *Schreibe eine Prozedur* `quadratischefunktion`*, die eine quadratische Funktion* $f(x) = ax^2 + bx + c$ *an einer Stelle auswertet.*
Dazu müssen der Prozedur 4 *Werte übergeben werden:* a*,* b*,* c *und* x
Nutze hierbei die bereits erklärte Prozedur `berechnequadrat`*.*
Berechne damit $f(x) = x^2 + 2x + 3$ *an der Stelle* $x = 0$*.*

7 Fallunterscheidung

Mit dem Schlüsselwort `ifelse` kann eine Fallunterscheidung getroffen werden. Als erstes wird die Frage übergeben. Diese Frage wird dann mit *ja* oder mit *nein* beantwortet. Wenn die Frage erfüllt ist, dann wird der zweite Parameter und sonst der dritte Parameter ausgeführt.

```
? ifelse (1<2) [print "ja] [print "nein]
ja
```

Die Anweisung läßt sich so verstehen: Wenn **(eins kleiner als 2)** ist, dann [**gebe ja aus**] und sonst [**gebe nein aus**].

```
? ifelse (3<=2) [print "ja] [print "nein]
nein
```

Hierbei wurde gefragt, ob 3 kleiner oder gleich 2 ist. Weitere Bedeutungen sind in folgender Tabelle aufgelistet:

Eingabe	**Bedeutung**
=	gleich
<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich

Das folgende Beispiel gibt das Maximum der beiden Parameter zurück:

```
to max :x :y
  ifelse (:x < :y) [output :y] [output :x]
end
```

So könnten Aufrufe aussehen:

```
? print max 4 5
5
? print max 6 3
6
```

```
? print max 6 max 30 -50
30
? print max max 6 30 -50
30
```

Aufgabe 7.1 *Schreibe ein Programm* `min`*, welches das Minimum zweier Zahlen zurückgibt!*

Aufgabe 7.2 *Schreibe eine Prozedur* `geradeungerade`*, welches auf den Bildschirm schreibt, ob eine Zahl gerade ist oder ungerade!*

Aufgabe 7.3 *Schreibe eine Prozedur* `loesequadratischegleichung`*, welches eine quadratische Gleichung löst!*
Dazu müssen 3 *Werte übergeben werden: a, b und c*

8 Rekursion

Man kann Prozeduren auch sich selber wieder aufrufen lassen – dies nennt sich Rekursion. Dabei ist natürlich darauf zu achten, dass dies nicht unendlich oft geschieht, sondern auch irgendwann endet.
Beispiel: Es soll das Kapital berechnet werden, wenn ein Anfangskapital k mehrere Jahre n bei gleichem Zinssatz p verzinst wird.

```
to zinseszins :k :n :p
  ifelse (:n > 0) [
    output zinseszins (:k*(1+:p)) (:n-1) :p
   ] [
    output :k
   ]
end
```

Damit kann nun bei einem Kapital von 100 berechnet werden, was man nach 10 Jahren bei 5 % $= 0.05$ hat:

```
? print zinseszins 100 10 0.05
162.889462677744
```

Aufgabe 8.1 *Schreibe eine Prozedur* `zinseszinsgewinn`*, die den Gewinn bei einer derartigen Geldanlage berechnet!*

Aufgabe 8.2 *Schreibe eine Prozedur* `potenz`*, die durch Rekursion die Potenz x^n für eine natürliche Zahl n berechnet!*

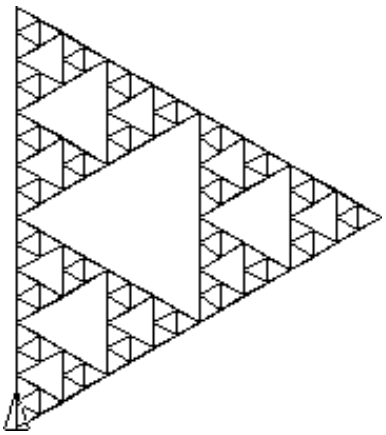
Beispiel: Sierpinski-Dreieck
Unter dem Sierpinski-Dreieck versteht man eine Figur, die dadurch entsteht, dass man

- ein gleichseitiges Dreieck nimmt,
- die Seitenlänge halbiert
- und damit neue Dreiecke in alle 3 Ecken zeichnet.

Dieser Prozeß wird dann beliebig oft wiederholt. Da wir das nicht unendlich oft machen können, müssen wir irgendwann abbrechen – beispielsweise sobald die Seitenlänge zu klein wird.

```
to sierpinski :s
  ifelse (:s > 20) [
    repeat 3 [sierpinski (:s/2) fd :s*2 rt 120]
   ] [
    repeat 3 [repeat 3 [fd :s rt 120] fd :s*2 rt 120]
   ]
end
```

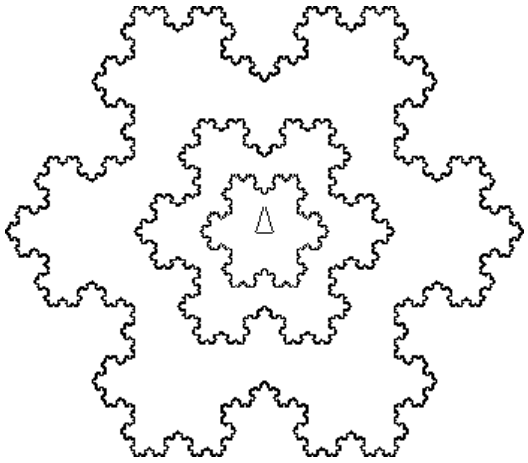
Der Aufruf `sierpinski 100` liefert folgendes Bild:



Aufgabe 8.3 *Schreibe eine Prozedur* `koch`*, die die Koch-Kurve zeichnet!*

Die Koch-Kurve entsteht aus einer Strecke indem man das mittlere Drittel der Strecke entfernt und stattdessen ein gleichseitiges Dreieck an gleiche Stelle zeichnet. Nun wiederholt man diesen Vorgang für jede Teilstrecke.

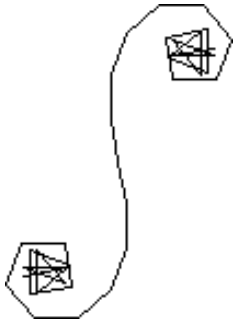
Aufgabe 8.4 *Schreibe eine Prozedur* `kochflocke`*, die* 3 *Koch-Kurven so zusammensetzt, dass eine Schneeflocke erkennbar wird.*



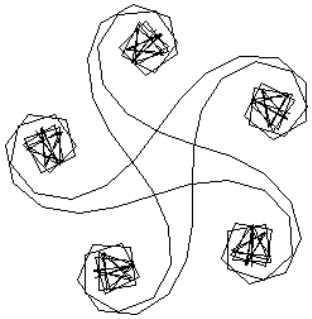
Man kann auf sehr einfache Weise durch Rekursion Muster erzeugen:

```
to spirale :s :w
  fd :s
  rt :w
  spirale :s (:w+10)
end
```

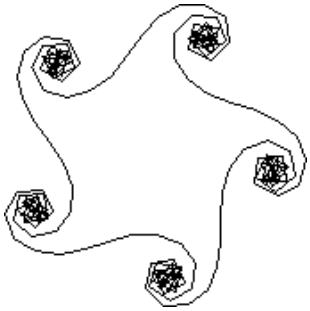
Beispiele:



`spirale 20 20`



`spirale 30 1`



`spirale 14 3`

9 Weitere Aufgabe

Aufgabe 9.1 *Schreibe eine Prozedur* `abs`*, die den Betrag einer Zahl berechnet!*

Aufgabe 9.2 *Schreibe eine Prozedur* `vorzeichen`*, die das Vorzeichen einer Zahl zurückgibt!*
Diese Prozedur soll also -1 *bei negativen Zahlen und sonst* 1 *zurückgeben.*

Aufgabe 9.3 *Schreibe eine Prozedur* `runden`*, die eine Zahl rundet!*

Aufgabe 9.4 *Schreibe eine Prozedur* `geometrischefolge`*, die das* n*-te Glied der geometrischen Folge* $\{a \cdot q^{n-1}\}_{n\in\mathbb{N}}$ *berechnet.*

Aufgabe 9.5 *Schreibe eine Prozedur* `geometrischepartialsumme`*, die die Summe der ersten* n *Glieder der geometrischen Folge* $\{a \cdot q^{n-1}\}_{n\in\mathbb{N}}$ *berechnet. Berechne damit die* 1000*-te Partialsumme der Folge* $\{\left(\frac{1}{2}\right)^{n-1}\}_{n\in\mathbb{N}}$.

Aufgabe 9.6 *Berechne näherungsweise eine Lösung der Gleichung*

$$x^n = a$$

Aufgabe 9.7 *Berechne näherungsweise eine Lösung der Gleichung*

$$a^x = b$$

10 Quellen, Literatur

- ucblogo: `http://http.cs.berkeley.edu/~bh/`
- wikipedia: `http://de.wikipedia.org/wiki/Logo_%28Programmiersprache%29`