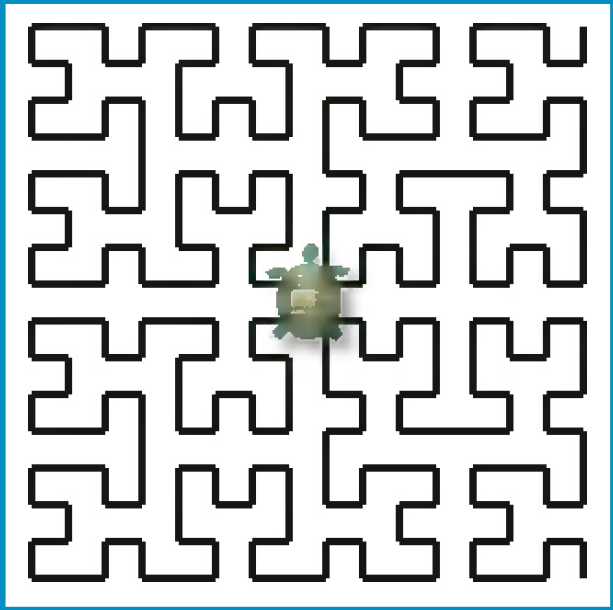
LERNEN MIT DIGITALEN MEDIEN

PROGRAMMIERUNG INTERAKTIVER GRAFIKEN

EINE EINFÜHRUNG MIT ACS LOGO



BAND 1: POLYGONE, SPIROLATERALE, REKURSIVE GRAFIKEN, L-SYSTEME

JOACHIM WEDEKIND

Mit der Reihe LERNEN MIT DIGITALEN MEDIEN stelle ich Texte zur Verfügung, die sich mit unterschiedlichen Aspekten des Lernens mit digitalen Medien befassen. Sie thematisieren praxisorientierte fach- und mediendidaktische Aspekte der digitalen Medien im Lehr-/Lernkontext.

IMPRESSUM:

Joachim Wedekind:
Programmierung interaktiver Grafiken: Eine Einführung mit ACS Logo. Band 1: Polygone, Spirolaterale, rekursive Grafiken, L-Systeme

Version 1.0

2014 Tübingen

Grafische Gestaltung und Satz: Joachim Wedekind
Das Werk steht elektronisch im Internet zur Verfügung:
http://programmieren.joachim-wedekind.de/logo/logo-buch/

LERNEN MIT DIGITALEN MEDIEN

PROGRAMMIERUNG INTERAKTIVER GRAFIKEN

EINE EINFÜHRUNG MIT ACS LOGO

BAND 1: POLYGONE, SPIROLATERALE, REKURSIVE GRAFIKEN, L-SYSTEME

JOACHIM WEDEKIND

Inhaltsverzeichnis

Vorwort

Wofür verwenden Sie eigentlich privat Ihren Computer? Noch vor nicht allzu langer Zeit wäre die Antwort vermutlich gewesen: Briefe schreiben, CDs katalogisieren oder Vergleichbares, vielleicht auch Spielen. Inzwischen ist sicherlich auch das Surfen im Internet hinzu gekommen. Sie verwenden dazu aufgabenbezogene Programme, die Sie unmittelbar nutzen können, also z.B. die Textverarbeitung oder den Internet-Browser. Weitergehende Möglichkeiten bieten Werkzeuge, die flexibel innerhalb eines Anwendungsbereichs einsetzbar sind. Vielleicht haben Sie ja mit einem Tabellenkalkulationsprogramm Ihre Einkommenssteuererklärung vorbereitet oder mit einem Webeditor Ihre persönliche HomePage gestaltet?

Die größte Flexibilität bieten Computer allerdings dann, wenn wir sie selber programmieren. Obwohl in gut sortierten Buchhandlungen dazu Regalmeter an Einführungen und Fortgeschrittenenliteratur zu finden ist, vermute ich doch, dass viele vor dieser sicherlich anspruchsvollsten Nutzungsform ihres Computers zurückschrecken. Es bleibt eben bei jeder Programmiersprache aufwendig und anspruchsvoll, sich die zugrunde liegenden Programmierkonzepte sowie die Unzahl (z. T. Hunderte davon) an Befehlen zu erarbeiten und angemessen für Problemlösungen einzusetzen.

Da sind dann schon etliche Schritte notwendig, um z. B. das beliebte und in vielen Einführungsbüchern genannte allererste Programm zu schreiben, das den berühmten Satz `Hello World` auf den Bildschirm zaubert. Die Entwicklungsumgebungen der derzeit gängigsten Programmiersprachen, wie C++, Java oder der Scriptsprachen, wie z.B. Python oder Ruby, sind außerordentlich leistungsfähig, aber dadurch eben auch sehr komplex und schwer erlernbar.

Einen deutlich leichteren Einstieg erlaubt dagegen die Programmiersprache Logo, die ursprünglich für Kinder und Schulen konzipiert wurde. Trotzdem ist diese Sprache durchaus leistungsfähig, für bestimmte Probleme sogar leistungsfähiger als andere. Logo erlaubt praktisch sofort und ohne Einstiegsschwelle loszulegen. Natürlich kann (und will) Logo nicht mit den oben genannten allgemeinen Programmiersprachen in puncto Leistungsfähigkeit und Anwendungsbreite konkurrieren, aber für das Anwendungsfeld interaktiver Grafiken ist es bestens geeignet, in mancher Hinsicht sogar einzigartig. Einige Beispiele in diesem Bändchen sollen das belegen.

Es gibt mathematische Phänomene, die durch ihre Visualisierung mit Hilfe des Computers große Popularität erlangten. Das gilt z.B. für das *Game of Life*, das John H. Conway 1970 vorgestellt hat, ebenso wie für das *Apfelmännchen*, das Mitte der 80er-Jahre als Visualisierung der Mandelbrot-Menge zum Sinnbild für das Forschungsfeld Chaos und Fraktale wurde. Wie viele andere ambitionierte Hobbyprogrammierer, habe auch ich das *Game of Life* nachprogrammiert (damals in FORTRAN). Bei der Beschäftigung mit weiteren mathematischen Konzepten, insbesondere den L-Systemen zur Modellierung natürlicher Pflanzenformen, habe ich bei deren programmtechnischer Umsetzung dann die Sprachelemente von Logo schätzen gelernt. Es hat großen Spaß gemacht, jeweils möglichst kompakte und gleichzeitig flexible Programme zu entwickeln, mit denen sich die Grafiken äußerst variabel darstellen ließen.

Mit diesem Büchlein möchte ich nun allen, die sich für die Programmierung und deren Anwendung bei der Erstellung interaktiver Grafiken interessieren, die Grundkonzepte und den Umgang mit Logo näher bringen. Dabei hoffe ich, ein wenig von der Faszination zu vermitteln, die ich

selbst bei der Erzeugung der Grafiken empfunden habe. Da ich weder Informatiker noch Mathematiker bin (sondern Unterrichtstechnologe und Mediendidaktiker), sollten Sie keine systematische Einführung in das Programmieren erwarten; auch bei den mathematischen Hintergründen muss ich auf die einschlägige Literatur verweisen (die Sie im Literaturverzeichnis finden). Ich verwende für die Umsetzung der Beispiele ACSLogo, eine Logo-Implementation von Alan C. Smith unter Mac OS X.

Zur Vorgeschichte dieses Büchleins: Ende der 80er Jahre arbeitete ich an einem Lehrerfortbildungsprojekt mit Studienmaterialien zum Thema "Lehren und Lernen mit dem Computer". Diese boten sowohl Orientierung zu allgemeinen pädagogischen Fragestellungen, die durch den Computereinsatz in der Schule aufgeworfen wurden, als auch praxisorientierte fach- und mediendidaktische Aspekte des Computereinsatzes im Fachunterricht. Ergänzt wurde dies durch erprobte Unterrichtseinheiten mitsamt der entsprechenden Software. Die 14 Studienbriefe mit Begleitsoftware fanden über mehrere Jahre viele Adressaten. In dieser Broschüre beziehe ich mich an einigen Stellen auf den Studienbrief „Schildkrötengrafik - Einfaches Programmieren von Grafiken“, den ich zusammen mit Hans Rauch verfasst und der dafür seinerseits auch die Software entwickelt hatte (Rauch & Wedekind, 1989). Sein Programm TURTLE-LLC war auf MS-DOS-Rechnern lauffähig und bot einen überschaubaren Befehlssatz, der sich im Wesentlichen auf den Grafikteil von Logo beschränkte.

Die hier verwendete Logo-Version ACSLogo ist dagegen eine vollständige und leistungsfähige Implementation der Sprache. Wenn Sie Spaß am Programmieren mit Logo gefunden haben, können Sie also weitergehende und anspruchsvollere Probleme als die von mir vorgestellten Beispiele in Angriff nehmen. Dabei wünsche ich Ihnen viel Freude und Erfolg.

Tübingen, Juli 2014 Joachim Wedekind

Einleitung

Diese Einführung in das Programmieren mit Logo soll Ihnen einen ersten, einfachen Einstieg in das selbständige Programmieren von Computern ermöglichen. Mit Hilfe der *Schildkrötengrafik* können Sie mit einem überschaubaren Befehlssatz zunächst einfache, aber dann zunehmend komplexe Grafiken programmieren. Sie lernen dabei die Formulierung von Algorithmen und ihre Codierung. Sie werden in einige zentrale Programmierkonzepte wie die Spracherweiterung durch Prozeduren, Kontrollstrukturen und Rekursion eingeführt. Die hierbei gewonnenen Erfahrungen können sicherlich auf moderne höhere Programmiersprachen übertragen werden. Mit ACSLogo kann man eigentlich gleich loslegen. Es braucht nicht viele Vorkenntnisse, um die ersten Erfolgserlebnisse zu erreichen. Wir werden deshalb auch nicht mit einer systematischen Spracheinführung beginnen, sondern direkt mit einfachen Beispielen einsteigen.

Zwei Kapitel werde ich allerdings voranstellen: ACSLogo steht in der Tradition der Logo-Bewegung. Die Philosophie von ACSLogo ist eigentlich nur aus dieser heraus zu verstehen. Ich skizziere deshalb im ersten Kapitel die Geschichte von Logo - auch wenn eilige Leser dies überspringen mögen. Im zweiten Kapitel geht es um Logo in der Schule, denn von Anfang an lagen die Beweggründe für die Entwicklung von Logo in dem Anspruch, Kindern und Jugendlichen einen neuen Zugang zum Problemlösen mit dem Computer zu eröffnen.

Mit ACSLogo arbeiten können Sie natürlich erst, wenn Sie es lauffähig auf Ihrem Rechner haben. Im Anhang A erfahren Sie alles Notwendige zur Installation des Programms. ACSLogo läuft nur auf dem Apple Macintosh (System MacOS X). Logo-Versionen für andere Plattformen (Windows, Linux) bzw. rein browserbasierte Versionen sind im Link-Verzeichnis aufgeführt.

Haben Sie ACSLogo erfolgreich installiert, fangen wir mit Beispielen an. Im Kapitel „Die ersten Schritte“ werden einerseits anhand der Turtle-Grafik grundlegende Programmierkonzepte (wie Erweiterbarkeit, Wiederholungen, Parameterübergabe u.a.) eingeführt. Andererseits lernen Sie dabei bereits einen Teil des Befehlssatzes von ACSLogo kennen.

In den darauf folgenden Kapiteln werden eine Reihe von Programmbeispielen besprochen, die einerseits Sprachkonzepte in der konkreten Anwendung zeigen, andererseits das Spektrum der grafischen Darstellungen verdeutlichen, die sich mit ACSLogo bearbeiten lassen. Diese Beispiele sollten Sie direkt am Rechner nachvollziehen, um auf diesen Erfahrungen aufbauend eigene Anwendungen zu entwickeln:

- Vielecke und Spiralen
- Spirolaterale
- Rekursive Grafiken
- L-Systeme

Weitere Anwendungen wie Punktwolken, Mosaike, Simulationen und Digitale Kunst sollen in einem zweiten Bändchen folgen.

Alles, was sich einer Darstellung anhand von beispielhaften Programmen entzieht, aber für Ihre weitere Arbeit mit ACSLogo unentbehrlich ist, wurde in die Anhänge verbannt. Sie finden dort also Installationshinweise (Anhang A) sowie die systematische Auflistung aller ACSLogo-Befehle (Anhang B).

ACSLogo ist zwar keine allgemeine Programmiersprache – zum Beispiel würden Sie sich schwer tun, mit ACSLogo einen Fragebogen samt Auswertungsroutinen zu erstellen. Dennoch ist ACSLogo durchaus leistungsfähig. Entsprechend viele Konzepte, Funktionen, Programmbefehle und ACSLogo-Eigenschaften gilt es zu erläutern. Eigentlich möchte man natürlich immer etliches davon schon zu Beginn und gleichzeitig kennen, um auch Anspruchsvolleres umzusetzen, was mit der linearen Darstellung im Buch aber schwierig zu bewerkstelligen ist.

Bevorzugt wird von mir ein handlungsorientiertes Vorgehen, d.h. das Erarbeiten der notwendigen Kenntnisse anhand praktischer Beispiele mit vorbereiteten Programmen.

Für unterschiedliche Textbestandteile verwende ich folgende Schriftdarstellungen:

- Web-Adresssen: http://joachim-wedekind.de (Unterstreichung)
- Programmquelltexte, Befehle und Programmausgaben: Schrift mit fester Zeichenbreite (`Courier`)
- Dateinamen: Schrift mit fester Zeichenbreite (`Courier`)
- Hinweise zu weiterführenden Aktivitäten: *Kursivschrift*
- Langzitate: *Kursivschrift*
- Elemente des Befehlssatzes von ACSLogo werden sukzessive immer dann neu eingeführt, wenn sie für die Umsetzung eines Algorithmus gebraucht werden. Sie finden ihre Beschreibung in grau unterlegten Kästen, auf die mit dem Schildkrötensymbol hingewiesen wird.

> **`Befehl`** *`Parameter`*
>
> Text zur Beschreibung des vorgestellten Befehls.

Hinweis: Die Logo-Programme (Prozeduren) sind im Text vollständig abgedruckt (durch Rahmen vom Fließtext abgehoben). Für alle, die sich das Eintippen ersparen wollen, können diese auch unter http://joachim-wedekind.de/Downloads/ACSLogoProgrammeBd1.zip komprimiert in einer ZIP-Datei heruntergeladen, entpackt und dann mit ACSLogo aufgerufen und ausgeführt werden.

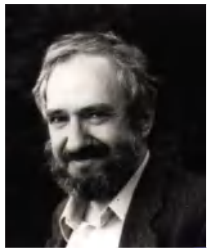
Ausblick: In diesem Band werden wir versuchen, die klassische Schildkrötengrafik mit ihrer *natürlichen Geometrie* (vgl. Hoppe, 1984, S. 24. f.) ein wenig auszureizen. Aus dem Mathematikunterricht sind wir allerdings ein anderes Koordinatensystem gewohnt, nämlich das *kartesische Koordinatensystem*, mit dem sich viele geometrische Sachverhalte am besten beschreiben lassen. Das lässt sich auch ohne große Umwege in Logo umsetzen und damit werden dann weitere Anwendungen realsierbar. Der zweite Band zur PROGRAMMIERUNG INTERAKTIVER GRAFIKEN macht sich das zu Nutze und wird Beispiele für Simulationen, Punktwolken und Digitale Kunst bringen.

Wie alles anfing ...

Es gibt sehr viele Programmiersprachen bzw. „Dialekte“ (wohl mehr als 2000!). Das macht allein schon deutlich, dass es vermutlich nicht die „beste“ Programmiersprache gibt. Welche verwendet wird, hängt von der Aufgabenstellung ab oder aber schlicht davon, welche Sprache gerade auf dem eigenen Rechner zur Verfügung steht - wobei Kostengründe dafür heute kaum noch eine Rolle spielen, denn für alle relevanten Sprachen gibt es absolut erschwingliche Versionen. Wie kann hier nun Logo eingeordnet und charakterisiert werden?

Logo kann nicht ohne einen Seitenblick auf die Schule und die dortige Diskussion über Programmiersprachen und ihre Rolle im Schulfach Informatik bzw. der informationstechnischen Grundbildung und deren Fachdidaktik betrachtet werden.

Die 1. Logo-Generation



Seymour Papert

Für die Anfänge von Logo müssen wir bis 1964 zurückgehen. In diesem Jahr kam der Mathematiker Seymour Papert an das berühmte MIT (Massachusetts Institute for Technology) in Boston. Zusammen mit Marvin Minsky – einem der Gurus der Forschung zur Künstlichen Intelligenz (KI) – gründete er das MIT Artificial Intelligence Laboratory. Ihre Forschung zu neuronalen Netzen beschrieben sie in dem Klassiker „Perceptrons“ (1969, Neuauflage 1988). Über den Kreis der KI-Fachleute hinaus bekannt geworden ist Papert aber durch die Entwicklung von Logo.

Papert hatte zuvor fünf Jahre bei Piaget in Genf gearbeitet und sich dort bei dem Entwicklungspsychologen intensiv damit beschäftigt, wie Kinder lernen. Mit diesem pädagogisch-psychologischen Hintergrund wollte er nun Werkzeuge entwickeln, mit denen Kindern das „Beste der Computerwissenschaften“ an die Hand gegeben werden sollte, damit sie ihre Denk- und Lernweisen verbessern können. 1967 erschien als Ergebnis seiner Überlegungen die erste Version von Logo, die unter der Leitung von Wallace Feuerzeig bei Bolt, Beranek and Newman entwickelt wurde.

Als Sprache für Kinder, aber eben keine „Spielzeugsprache“, war Logo (als Abkömmling der listenorientierten KI-Sprache LISP) modular, erweiterbar und interaktiv konzipiert. Ihre Charakterisierung mit „no treshold, no ceiling“ (keine Einstiegshürde, keine Begrenzung nach oben) soll andeuten, dass Kinder mit Logo sehr rasch problemorientiert arbeiten können, dass die Sprache aber mächtig genug ist, Lernende an komplexe Probleme heranzuführen.

Besonders die Listen zur Verarbeitung von Symbolen und komplexen Datenstrukturen haben sich dafür als hilfreich erwiesen, denn Listen sind als rekursiver Datentyp besonders dafür geeignet, verschachtelte Strukturen darzustellen. Das kann dann bis zu Fragen der KI auf Hochschulniveau gehen (so beschrieben in dem Lehrbuch von Bundy, 1980; deutsche Fassung 1986).

Bodenturtle

Als zentrale Komponente von Logo wurde ab 1970 die Turtlegrafik (Schildkrötengrafik)[1] eingeführt, durch die mit einfachen Grundbefehlen ansprechende und komplexe Grafiken erzeugt werden können. Grafische Ausgabegeräte waren zur Zeit der Entwicklung von Logo noch sehr teuer. Es wurde deshalb anfangs ein kleiner schildkrötenähnlicher Roboter eingesetzt: die Turtle. Dieses kleine Gefährt konnte sich vorwärts bewegen und drehen. Ein Schreibstift konnte angehoben und gesenkt werden, so dass durch die Bewegungen der Schildkröte auf einem darunter liegenden Papier Linien gezogen wurden. Mit der Verfügbarkeit immer billigerer Computer wurden allerdings gerade diese Roboter zum limitierenden Kostenfaktor und bald wurde so die mechanische durch eine elektronische Schildkröte auf einem Bildschirm abgelöst; sie wurde zu einem kleinen Symbol mit Richtungsanzeiger.

MIT Logo für den Apple II

Die Verbreitung von Logo begann denn auch richtig in den späten Siebzigern mit den ersten erschwinglichen Personalcomputern. Die ersten entsprechenden Logo-Versionen gab es für die 8-bit Rechner Apple II (Apple Logo) und Texas Instrument TI 99/4 (TI Logo). Ab 1980 wurden etliche größere Schulversuche mit Logo durchgeführt, so das Lamplighter School Project in Dallas, Texas, oder das Computers in Schools Project in New York City.

Auch in der Bundesrepublik wurde mit Logo an Schulen experimentiert und gearbeitet, u.a. schon von 1974 bis 1979 in dem Projekt PROKOP (Böcker, Fischer & Plehnert, 1986; Böcker, Fischer, & Schollwöck 1987) in der Sekundarstufe I und II. Seit 1980 hat sich dann die Arbeitsgruppe um Herbert Löthe an der Pädagogischen Hochschule Esslingen, später Ludwigsburg, Verdienste um die Erprobung in der Primar- und Sekundarstufe I erworben. Von dort wurde auch die Eindeutschung von Logo (mit deutschen Schlüsselworten und Fehlermeldungen) betrieben. Einige dieser Arbeiten sind bis heute im Internet verfügbar.

Der Name Logo stand bald nicht nur für die Programmiersprache, sondern auch für prägnante (aber insbesondere in der deutschen Mathematikdidaktik nicht unumstrittene) pädagogische Vorstellungen von interaktiven Lernumgebungen. So schreibt Harald Abelson (1983, S. VII), einer der Mitarbeiter Paperts:

> *„Die Sprachen der Logo-Familie sind gezielt so entwickelt, dass sie die Computer zu flexiblen Hilfsmitteln machen, die das Lernen, das Spielen und das Erforschen unterstützen. Wir Wissenschaftler, die wir an Logo arbeiten, lassen uns von der Vision eines pädagogischen Werkzeugs leiten, das zugleich ohne*

[1] In den deutschsprachigen Logo-Einführungen (z.B. Abelson, 1983; Hoppe & Löthe, 1984 oder Ziegenbalg, 1985) und in den deutschen Versionen der Programmiersprache Logo (aktuell in MSWLogo) wird statt Schildkröte häufig der Ausdruck Igel (wegen der Kürze des Wortes) verwendet.

Einstiegsschwelle und ohne Begrenzung nach oben ist. Wir versuchen also, selbst sehr jungen Schülern eine selbständige und selbstbestimmte Beherrschung des Computers zu ermöglichen (...). Zugleich meinen wir, dass Logo ein Allzweckprogrammiersystem sein sollte, das beachtliche Ausdrucksfähigkeit und umfangreiche Formulierungsmöglichkeiten hat. Es ist tatsächlich so, dass wir diese beiden Ziele eher als Ergänzung denn als Widerspruch betrachten."

In seinem Buch „Mindstorms: Kinder, Computer und Neues Lernen" hat Papert (1982) seine Überlegungen zusammengefasst. Es ist dabei weniger ein Buch über Logo, sondern über Paperts Antrieb, Kindern eine Art Mathematikland zu schaffen, in dem sie Mathematik lernen können sollen, so wie eine Sprache am besten durch Aufenthalt in dem entsprechenden Land gelernt wird. Er leitet unter Berufung auf Rousseau und Piaget ein didaktisches und bildungstheoretisches Modell ab, mit dem gesellschaftliche Anforderungen und Verpflichtungen mit den Bedürfnissen der Kinder in Einklang gebracht werden sollen. Eine Lektüre, die immer noch lohnt[2].

Jedenfalls, als Anfang der 80er-Jahre noch um die „richtige" Programmiersprache für den schulischen Informatikunterricht gestritten wurde, gehörte Logo zu den „schulspezifischen" Sprachen, die in Modellversuchen erprobt wurden. Es stand damit in Konkurrenz zu ELAN, PASCAL-E, BASIC oder COMAL-80 (vgl. das Themenheft „Programmiersprachen" der Zeitschrift LOGIN 3/1983). Für Logo sprach, dass es als interaktive, listenverarbeitende, prozedurale Programmiersprache durch die lernpsychologisch motivierten Eigenschaften didaktische Grundideen tragen kann (Tauber, a.a.O., S. 37):

- die Unterstützung von entdeckendem, spielerischem Lernen
- das Formulieren und Ausprobieren von Vermutungen
- das Lernen aus Fehlern und deren Korrektur
- die Zerlegung von Problemen in Teilprobleme und deren Lösung mit Hilfe von Prozeduren
- das Baukastenprinzip, d.h. die Erweiterung der Sprache durch eigene problemspezifische Prozeduren
- die Erarbeitung von programmiersprachlichen Konzepten mit der Schildkrötengrafik
- das einfache interaktive Arbeiten mit sofortigem grafischem Feedback

Logo erlebte in dieser Zeit einen kleinen Boom. Zahlreiche Versionen wurden für unterschiedliche Rechner entwickelt, da es ja noch nicht die heutige Monokultur bei den Betriebssystemen gab. So das Terrapin Logo für Apple II und Commodore 64 , sowie von Logo Computer Systems, Inc. (LCSI, eine von Papert gegründete Firma) Versionen für verschiedene Rechnertypen und in mehr als einem Dutzend Sprachen.

Die 2. Logo-Generation

Mitte der 80er Jahre wurde Logo mit neuen Funktionalitäten weiter entwickelt. Es entstanden das IBM Logo, Versionen für den Atari ST und MacLogo für den Apple Macintosh. Für den Macintosh wurde von Coral Software mit ObjectLogo eine objektorientierte Version vorgestellt. Dennoch ist Logo bei Anwendungsprogrammierern nie populär geworden, obwohl sich nun sogar stand-alone Anwendungen mit höherer Ablaufgeschwindigkeit erstellen ließen.

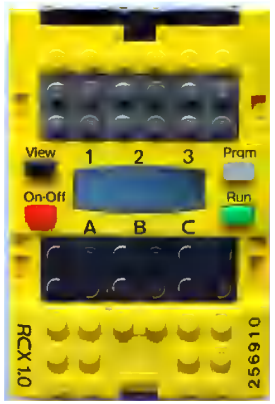
[2] Wichtige Aufsätze von Seymour Papert sind nach wie vor zugänglich unter http://www.papert.org/works.html

1985 erschien mit LogoWriter (von LCSI) zum ersten Mal eine Version, bei der mehrere Turtles gleichzeitig über den Bildschirm bewegt werden konnten. Außerdem besaß es eine integrierte Textverarbeitung. Es wurde in vielen Sprachen implementiert und fand weltweit Verbreitung. Zwar verlor Logo Mitte der 90er Jahre in den USA an den Schulen an Popularität, aber vor allem in Südamerika wurde es breit an den Schulen eingeführt. In Großbritannien wurde es sogar im nationalen Curriculum verankert.

An der Universität Bratislava wurde Comenius Logo entwickelt und 1993 vorgestellt, das unter dem Namen SuperLogo in mehreren Sprachen weite Verbreitung gefunden hat und heute als ImagineLogo von Logotron vertrieben wird. Mit MicroWorlds wird auch von LCSI weiterhin eine aktuelle Version für Windows und Macintosh angeboten. Diese modernen Logo-Implementationen haben Multi-Tasking-Fähigkeiten, d.h. sie erlauben den gleichzeitigen Ablauf mehrerer Prozesse. Das bedeutet z.B., dass mehreren Schildkröten (bzw. allgemeiner: grafischen Objekten) gleichzeitig Befehlsfolgen zugeordnet und sie unabhängig voneinander aktiviert werden können. Dadurch können sehr leicht auch aufwendigere Animationen realisiert werden.

Insgesamt gibt es also eine große Zahl an Logo-Versionen, inzwischen auch für Tablets (wie z.B. LogoPlus oder Logo Draw, diese dann aber meist beschränkt auf die Turtle-Grafik) oder rein browserbasiert, wie z.B. jslogo. Im Logo Tree Project (Boytchev, Rev. 2.09, 2014) sind über 290 Versionen aufgelistet; allerdings werden davon aktuell nur etwa 20 aktiv weiter entwickelt. Die Vielfalt der Versionen hat zu einer Aufsplitterung der Nutzergruppen von Logo geführt. Es gibt keinen Sprachstandard und die Versionen unterscheiden sich z.T. in Sprachumfang und Syntax erheblich. Die Schildkrötengrafik ist letztlich der einzige gemeinsame Nenner. Gerade bei den kommerziellen Systemen ist das Verständnis von Logo als Werkzeug zum Bearbeiten und Verstehen komplexer Probleme in den Hintergrund geraten (Chakraborty, Graebner & Stocky, 1999).

Es gibt aber nach wie vor Logo-Versionen, die für den Einsatz in der Schule geeignet sind (so wie das hier vorgestellte ACSLogo) und die weiter gepflegt werden. Ebenso gibt es nach wie vor deutschsprachige Logo-Versionen, etwa von Gerhard Otte (für Windows; besonders interessant, weil u.a. mit Treibern für Fischer-Technik-Modelle) sowie unterschiedliche deutschsprachige MSW-Logo Versionen der PH Ludwigsburg (an der auch die früheren deutschen Logo-Versionen entstanden sind), dabei eine Version mit 3D-Grafikbefehlen.



Der RCX Baustein, das Herzstück von LEGO/Logo

Inzwischen bewegen sich die Schildkröten auch wieder auf dem Boden und nicht nur auf dem Bildschirm. Mit LEGO/Logo wurde 1988 von Mitchel Resnick mit Stephen Ocko (vom MIT Media Lab) ein System vorgestellt, bei dem mit Lego-Bausteinen Maschinen gebaut werden können, die mit Motoren und Sensoren ausgestattet sind. Über ein spezielles Interface können diese Maschinen mit Hilfe von Logo-Befehlen gesteuert werden. Es stellt damit einen Vorläufer der populären Mindstorm-Serie dar. Diese Hard-Softwarekombination ist an Schulen recht verbreitet und es gibt viele spannende Unterrichtsbeispiele. In der FIRST LEGO League, einem

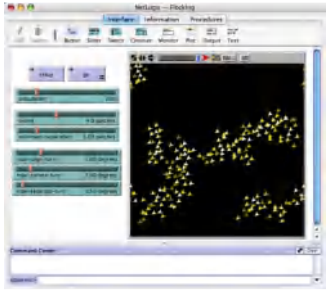
jährlichen Roboterwettbewerb, können Kinder und Jugendliche zwischen zehn und 16 Jahren innerhalb eines Teams zu einem vorgegebenen Thema einen vollautomatischen Roboter planen, programmieren und testen, um damit eine knifflige "Mission" zu meistern und eine wissenschaftliche Aufgabe zu bearbeiten.

Die 3. Logo-Generation: Agenten statt Schildkröten

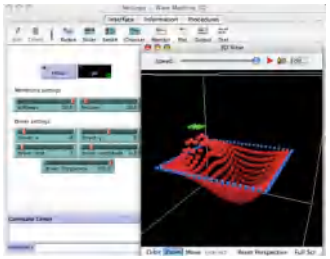
Logo mit mehreren Turtles bietet bereits spezielle Möglichkeiten. Seit Mitte der 90er Jahre gibt es nun neue Versionen, mit denen nicht nur einige wenige sondern Tausende von Turtles gleichzeitig gesteuert werden können. StarLogo war die erste dieser leistungsfähigen Versionen, entwickelt im Media Laboratory des MIT. Wie bisher auch lassen sich die Turtles mit Befehlen steuern. Zusätzlich kann aber auch die Umgebung der Turtles definiert werden und die Turtles können untereinander und mit der Umwelt interagieren. Allgemein gesprochen handelt es sich dabei um Multi-Agenten Systeme. Allgemeine Systeme für agentenbasierte Simulationen sind etwa SWARM, Repast oder EcoLab.

Die Programmierumgebungen der StarLogo-Familie sind speziell für unterrichtliche Bedürfnisse entwickelt worden (Resnick, 1994). So wie die klassischen Logo-Versionen prozedurales Denken und die Rekursion für den schulischen Unterricht erschlossen haben, gilt dies nun für die massive Parallelität agentenbasierter Simulationen. Dabei geht es dann aber nicht nur um ein Revival von Logo als Programmiersprache für die Schule, sondern um den schülergerechten Zugang zur Modellierung und Simulation dynamischer Systeme. Die Programmierumgebungen erlauben die schnelle Implementation unterschiedlicher Systeme durch die Lehrerinnen und Lehrer und gegebenenfalls auch durch die Schülerinnen und Schüler selbst. Die Erweiterungen betreffen im wesentlichen drei Aspekte:

- Es handelt sich um massiv parallele Sprachen – so können Tausende von Turtles zur gleichen Zeit Aktionen ausführen. Dies ist für die Untersuchung komplexer dynamischer Systeme eine Grundvoraussetzung.
- Die Turtles besitzen „Sensoren“. Sie können z.B. andere Turtles in ihrer Nähe entdecken und unterscheiden, sie können Eigenschaften ihrer Umwelt erkennen und sie können Gradienten folgen. Solche Turtle-Turtle und Turtle-Umwelt-Interaktionen sind grundlegend für das Experimentieren mit selbst organisierenden Phänomenen.
- Die Turtles bewegen sich in einer definierbaren Umwelt. Diese Umwelt ist in kleine quadratische Felder unterteilt, denen vergleichbare Eigenschaften wie den Turtles zugewiesen werden können, die sich im Gegensatz zu diesen allerdings nicht bewegen können. Damit wird u.a. indirekte Kommunikation zwischen den Turtles möglich, wenn diese bestimmte „Marker“ auf den Feldern hinterlassen, die von anderen Turtles ausgewertet werden können.



Simulation des Vogelflugs in NetLogo



3D Version einer Wellenmaschinensimulation

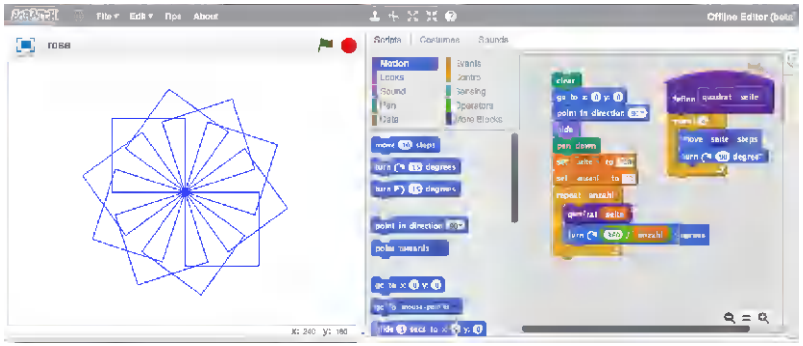
Die leistungsfähigste Variante ist NetLogo, das von der Arbeitsgruppe um Professor Uri Wilensky (Center for Connected Learning and Computer-Based Modeling, Northwestern University Evanston) entwickelt wurde. In seiner Arbeitsgruppe wurden zahlreiche Anwendungen für verschiedene Fächer entwickelt (vgl. z.B. Stieff & Wilensky, 2002; zum Download-Paket gehört eine umfangreiche Modellbibliothek für mehrere Fächer). NetLogo wird sehr gut gepflegt und weiter entwickelt: In einer 3D-Version von NetLogo können sich die Turtles in drei Dimensionen bewegen, was zu besonders aussagekräftigen Animationen führt. Schließlich ist in NetLogo inzwischen ein grafischer Editor integriert, mit dem Modelle in System Dynamics Repräsentation aufgebaut werden können und damit Makro- und Mikrosimulationen in einem System berechenbar sind ... aber das ist eine andere Geschichte (oder ein weiteres Büchlein).

Zu erwähnen ist noch, dass die Schildkrötengrafik auch in anderen Programmiersprachen Eingang gefunden hat. Unter anderem in Squeak, einer frei verfügbaren Smalltalk-Version (Stéphane Ducasse, 2005, hat ein eigenes Buch darüber geschrieben). Squeak und auch Logo (als Turtle Art) sind u.a. Bestandteil der Software-Ausstattung des XO-Laptops (auch als 100$-Laptop bekannt geworden).

Die 4. Generation: Logo-Revival mit visueller Programmierung?

Mit einer Reihe visueller Programmierumgebungen erleben wir momentan fast so etwas wie ein Logo-Revival, dessen Ende derzeit noch nicht abzusehen ist. Begonnen hat dies 2007 mit Scratch (entwickelt am MIT Media Lab unter Leitung von Mitchel Resnick). Programme werden dabei aus Bausteinen zusammengesetzt, die zumeist allein durch ihre Form nur das Einklinken passender Elemente erlauben und dadurch Syntax-Fehler vermeiden helfen. Daran knüpfen z.B. Turtle-Art, blockly oder MIT App Inventor an.

Ein interessante Weiterentwicklung ist Snap! mit mächtigen Optionen, u.a. um eigene Kontrollstrukturen zu definieren. Da es HTML5 und Javascript verwendet, läuft es vollständig im Webbrowser - auch auf mobilen Geräten.



visuelle Programmierumgebung Scratch

Logo in der Schule

Wie wir gesehen haben, wurde Logo als Sprache für Kinder entwickelt. Auch wenn dies kein Lehrbuch sein soll und die Beispiele nicht als Materialien für konkrete Unterrichtseinheiten gedacht sind (dazu kann auf Hromkovič, 2010, verwiesen werden), soll in diesem Kapitel doch eine Positionierung im schulischen Kontext versucht werden.

Unter dem Stichwort *Programmieren für Alle* ist eine Diskussion (wieder) aufgeflammt (u.a. angestossen von Douglas Rushkoff mit seinem Buch *Program or be Programmed*, 2011), wann und mit welchen Werkzeugen Grundkonzepte der Informatik vermittelt werden sollen. Dabei ist wohl weniger entscheidend, welche Programmiersprache eingesetzt wird. Den Anspruch kindgerechter Einführungen erheben z. B. Bücher zu Python (Briggs, 2013), Java (Eshel, 2011) oder Java Script (Strom, 2014). Wichtiger ist die Vermittlung zentraler Begriffe bzw. Konzepte der Informatik, wie *Algorithmus* und *Programm*.

Computer sind Maschinen, die wir zur Erledigung bestimmter Aufgaben einsetzen können. Sie unterscheiden sich von den meisten anderen Maschinen insbesondere dadurch, dass die eigentlichen Maschinen (die Hardware) erst durch Programme und Daten (die Software) in ihrem Arbeitsablauf bestimmt und damit universell einsetzbar werden. Der Computer "tut" nichts von selber, sondern wir müssen stets die entsprechenden Anweisungen in einer von der Maschine verwertbaren Form eingeben: als Programm.

Programme sind immer in einer Programmiersprache geschrieben. Allerdings "versteht" jeder Computer letztlich nur seine eigene Maschinensprache. Maschinensprachen sind äußerst unanschaulich und eigentlich nur für Spezialisten verständlich. Das Programmieren in Maschinensprache ist zeitaufwendig, umständlich und fehleranfällig. Für den Nicht-Spezialisten und zugeschnitten auf bestimmte Anwendungsfelder wurden deshalb höhere Programmiersprachen entwickelt, die Formulierungen zulassen, die den üblichen sprachlichen Darstellungsweisen wesentlich näher liegen.

Auch wenn uns eine solche höhere Programmiersprache zur Verfügung steht, kommen wir um wesentliche vorbereitende Arbeitsschritte nicht herum. Am Anfang steht das Problem, bei dessen Lösung wir uns vom Computer Hilfe erwarten. In der Denkpsychologie wird dann von einem *Problem* gesprochen, wenn wir von einem unerwünschten Anfangszustand zu einem erwünschten Zielzustand gelangen wollen, aber (zumindest augenblicklich) nicht über die Mittel verfügen, um den unerwünschten in den erwünschten Zustand zu überführen. Sind die Mittel im Prinzip bekannt und anwendbar, wird dagegen von einer *Aufgabe* gesprochen (Dörner, 1979, S. 10). So gesehen können wir dem Computer prinzipiell immer nur die Bearbeitung von Aufgaben übertragen.

Wir haben also diejenigen Teile der Problemstellung festzustellen, die wir als Aufgaben formulieren können, weil uns die Mittel und Wege bekannt sind, sie zu lösen. Wir können damit eine Abfolge eindeutig formulierter Anweisungen definieren, die die Lösung der Aufgabe bewirken. Wir sprechen dann von einem *Algorithmus*. Ist die Formulierung von Algorithmen möglich, sind auch die Chancen gestiegen, den Computer als Hilfsmittel nutzen zu können. Allerdings gibt es genügend Aufgaben, die sich der Abarbeitung durch den Computer entziehen.

Beim Programmieren, d.h. der Nutzung des Computers als Werkzeug zum Problemlösen, können wir (nach Ziegenbalg, 1984) stark vergröbernd drei Charaktere unterscheiden. Der erste Typ, der als *Drauflosprogrammierer* bezeichnet werden könnte, beginnt sofort ein Programm zu schreiben, kaum dass das Problem formuliert ist. Der zweite Typ, der als *Systematiker* bezeichnet werden könnte, hält nichts davon, sofort am Computer loszulegen, sondern er geht das Problem systematisch an. Der dritte Typ könnte schließlich als planend, gleichzeitig experimentierend vorgehender *Ingenieur* bezeichnet werden. Jeder dieser typisierten Charaktere kann durch bestimmte Programmierumgebungen gefördert oder gar provoziert werden. Durch den ersten Typ wird ein häufig mit BASIC-Systemen verbundener Arbeitsstil beschrieben. Der zweite Typ ist eher charakteristisch für das Arbeiten in Pascal-Systemen. Andere Programmierumgebungen – wie eben die Logo-Systeme – unterstützen dagegen durchaus die Arbeitsweise des dritten Typs.

Logo erlaubt einen deutlich leichteren Einstieg als das mit anderen gängigen Programmiersprachen möglich wäre. Deren Leistungsfähigkeit korrespondiert mit hoher Komplexität und schwerer Erlenbarkeit. Es gibt damit es aus meiner Sicht nach wie vor gute Gründe, in der Schule mit Logo zu arbeiten. Das Schweizer Projekt PrimaLogo empfieht Logo ausdrücklich für das Programmieren an Grundschulen[3].

Interaktivität: Die Interaktivität von Programmierumgebungen soll den Anwendern einen möglichst unmittelbaren Zugang zu allen Funktionen erlauben. Als interaktive Systeme stellen alle Logo Versionen (auch ACSLogo) deshalb Kommandos zur Verfügung, die auf Betriebssystemebene wirken, um einen Wechsel zwischen den verschiedenen Ebenen zu vermeiden.

Einfachheit: Bei der Arbeit wird zwischen Direkt- und Programmiermodus unterschieden. Im Direktmodus wird ein(e) Befehl(sfolge) sofort analysiert und ausgeführt. Im Programmiermodus werden Befehlsfolgen beim Verlassen des integrierten Editors auf ihre syntaktische Korrektheit untersucht. Somit ist ein leichter Übergang zwischen der Phase des Eingebens und der Programmausführung gegeben.

Erweiterbarkeit: Der Sprachumfang des Systems kann durch die Zusammenfassung mehrerer einzelner Befehle erweitert werden. In der Informatik wird dann von Prozeduren (mit Namen versehenen Befehlsfolgen) gesprochen.

Parameterübergabe: Prozeduren können – ebenso wie den meisten Grundworten – Zahlenwerte übergeben werden.

Rekursion: Als ein sehr mächtiges Konzept hat sich in der Informatik die Rekursion erwiesen: Eine Prozedur kann sich selber aufrufen. Dieses Konzept erlaubt oft verblüffend einfache Lösungen für komplexe Fragestellungen, ist aber nicht ganz einfach zu verstehen.

Listenverarbeitung: Listen sind eine geordnete Zusammenfassung von einzelnen Datenobjekten zu einem neuen Datenobjekt. Als Listenelemente können Wörter, Zahlen oder wiederum Listen fungieren, auf die Standardprozeduren angewandt werden können.

Vollständigkeit: Zum Schreiben von Programmen oder Programmteilen steht ein integrierter Editor zur Verfügung (eine sog. integrierte Entwicklungsumgebung).

[3] PrimaLogo - Programmieren an Primarschulen: http://primalogo.ch

Automatische Formatierung: Alle Logo-Versionen besitzen einen integrierten Editor, in dem die Programmzeilen teilweise automatisch formatiert werden. Die optische (farbliche) Gestaltung erleichtert das Verständnis der Programmstrukturen.

Fehlermeldungen: Werden im Direkt- oder Programmiermodus bei der Überprüfung der Befehle syntaktische Fehler festgestellt, so wird eine kontextabhängige Fehlermeldung zurückgegeben, mit Markierung der Fehlerstelle innerhalb des Codes.

Hilfen: Es steht optional eine ausführliche Hilfefunktion zur Verfügung.

Hinzu kommt, dass Logo reich ist an Metaphern, wodurch es für den Einsatz bereits an der Grundschule besonders geeignet ist (Kohler, Spannagel & Klaudt, 2008):

1. Es gibt Objekte, z.B. kybernetische Tiere (die Schildkröte).
2. Diese haben Fähigkeiten und können etwas tun (Prozeduren/Funktionen).
3. Sie haben ein Wissen über sich und ihre Umgebung (Datenhaltung).
4. Man kann Aufträge bzw. Befehle an sie senden (Prozeduren).
5. Man kann Anfragen stellen, die sie beantworten (Funktionen).
6. Man kann für sie eine Umgebung (Mikrowelt) entwickeln und ihr Verhalten verändern (Programmieren)

Ein Vorzug von Logo ist weiterhin, dass es nicht nur leicht erlernbar ist, sondern auch den Übergnag zu sehr komplexen, für die Informatik zentralen Fragestellungen erlaubt. Brian Harvey zeigt das eindrucksvoll in seinen drei Bänden *Computer Science Logo Style* (Harvey, 1997). Er zeigt u.a. wie mit Logo Parser und Compiler für BASIC geschrieben oder Fragen der Künstlichen Intelligenz bearbeitet werden können.

Zugegebenermaßen sind die Anwendungen in den folgenden Kapiteln von solchen Ansprüchen weit entfernt. Dennoch ist ACSLogo gut geeignet, die Logo-Konzepte auf interessante Problemstellungen anzuwenden und ansprechende Grafiken zu programmieren. Darauf wollen wir uns in den folgenden Kapiteln konzentrieren[4].

[4] Eine breiter angelegte Einführung in ACSLogo bietet der Programmautor Alan C. Smith in seinem Tutorial, das nach Programmstart über das Hilfe-Menü zugänglich ist.

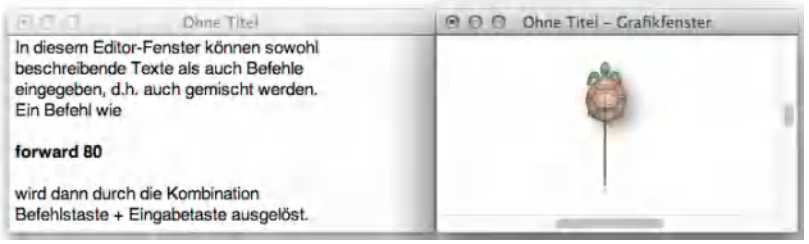
Die ersten Schritte

ACSLogo bietet sich dazu an, einfach mal loszulegen und so erste Erfahrungen mit der Schildkrötengrafik zu gewinnen. Dazu können Sie zunächst einfache Befehlsfolgen im *Direktmodus* verwenden. Sie werden diese dann später im *Programmiermodus* wieder finden.

Die Beispiele in diesem Kapitel sollen erste Erfahrungen im Umgang mit der Schildkrötengrafik vermitteln. Die Beispiele dienen einer Einführung in die Konzepte

- der Erweiterbarkeit,
- der Wiederholung,
- der Parameterverwendung,
- der Zerlegung in Teilaufgaben und
- der Rekursion.

Wir gehen im folgenden davon aus, dass Sie ACSLogo gestartet haben (vgl. Anhang A). Das Programm zeigt dann das Grafikfenster und bietet in einem zweiten Fenster einen Editor für den Direktmodus an.



Die Befehle oder Befehlsfolgen der folgenden Beispiele sollten Sie immer direkt am Computer ausprobieren und übungshalber abwandeln und erweitern!

Der Direktmodus

Im Editor-Fenster können wir einzelne Befehle[5] bzw. Befehlsfolgen eingeben, die dann sofort ausgeführt werden. Im Grafikfenster werden die Bewegungen der Schildkröte dargestellt und wir können dort die Auswirkungen der jeweiligen Aktionen verfolgen: Die Schildkröte kann durch Eingabe der Befehlsworte `Forward`, `Back`, `Left` und `Right` auf dem Bildschirm unter Hinterlassen einer Spur bewegt werden. Die Befehle werden dazu einschließlich dem Zahlenwert

[5] Die jeweils benötigten Grundbefehle von ACSLogo werden in den grau unterlegten Kästchen erläutert. Eine vollständige Befehlsübersicht (ca. 150 Befehle) findet sich in Anhang B. Eine genaue (englischsprachige) Erläuterung aller verfügbaren Befehle mit Beispielen gibt es in Form der *Logo Command Reference* in der Datei `LogoCommandRef.pdf`, die im Download von ACSLogo mit enthalten ist. Eine systematische Einführung verbietet sich hier aus Platzgründen. Die Befehle beziehen sich z.B. auf die Darstellung im Grafikfenster, die Behandlung von Listen bzw. Zeichenketten, auf mathematische Operationen und Funktionen, die Kontrollstrukturen oder die Ein-/Ausgabe. Die folgenden Programmbeispiele sind also als Anregung gedacht – die vorgestellten Konzepte und Grundworte können und sollen in eigenen Erweiterungen und Anwendungen unter Hinzuziehen des Anhangs B vertieft werden.

für eine gewünschte Strecke bzw. einen gewünschten Winkel eingegeben. Das Zeichnen eines Quadrats kann also beispielsweise mit folgendem Befehl begonnen werden:

```
Forward 80
```

WICHTIG: Zur Ausführung des Befehls muss sich die Schreibmarke vor dem Befehl, innerhalb oder am Ende der Befehlszeile befinden. Durch Drücken der <Befehlstaste> und <Return> (⌘+←) wird der Befehl bzw. alle Befehle einer Befehlszeile ausgeführt! Mit der Eingabe von <Return> springt die Schreibmarke immer in eine neue Zeile und es kann ein weiterer Befehl oder Text eingegeben werden.

Die Schildkröte bewegt sich damit 80 Bildschirmpunkte (Pixel) in ihre Blickrichtung (d.h. also nach Programmstart nach oben). Die Blick- bzw. Bewegungsrichtung der Schildkröte kann durch die Befehle `Right` bzw. `Left` geändert werden. Wir geben deshalb ein (der Abschluss jeder Eingabe durch <*Return*> wird im folgenden nicht mehr zugefügt!):

```
Right 90
```

Es ist einleuchtend, wie nun ein vollständiges Quadrat erhalten werden kann. Wir geben nacheinander die folgenden Befehle[6] ein

`Forward` *`Zahl`*

Mit dem Befehl `Forward` ***`Zahl`*** (abgekürzt `fd` ***`Zahl`***) bewegt sich die Schildkröte um die entsprechende Zahl von Bildpunkten in die Richtung, in die sie aktuell zeigt. Falls der Zeichenstift abgesenkt ist, wird bei dieser Bewegung eine Strecke gezeichnet. Ansonsten wird nur die Schildkröte weiter bewegt.

`Back` *`Zahl`*

Mit dem Befehl `Back` ***`Zahl`*** (abgekürzt `bk` ***`Zahl`***) bewegt sich die Schildkröte um die entsprechende Zahl von Bildpunkten in die Richtung, in die sie aktuell zeigt (mit dem Befehl `back` steht also die Umkehroperation zu dem Befehl `Forward` zur Verfügung. Dabei gilt: `Forward -`***`Zahl`*** ist gleichbedeutend mit `Back` ***`Zahl`***!).

`Right` *`Zahl`*

Mit dem Befehl `Right` ***`Zahl`*** (abgekürzt `rt` ***`Zahl`***) wird die Schildkröte relativ zur aktuellen Richtung um den durch ***`Zahl`*** angegebenen Winkelwert (in Grad) im Uhrzeigersinn gedreht.

`Left` *`Zahl`*

Mit dem Befehl `Left` ***`Zahl`*** (abgekürzt `lt` ***`Zahl`***) wird die Schildkröte relativ zur aktuellen Richtung um den durch ***`Zahl`*** angegebenen Winkelwert (in Grad) entgegen dem Uhrzeigersinn gedreht (mit dem Befehl `back` steht also die Umkehroperation zu dem Befehl `Right` zur Verfügung. Dabei gilt: `Right -`***`Zahl`*** ist gleichbedeutend mit `Left` ***`Zahl`***!).

[6] Für einige, aber nicht für alle Befehle gibt es Abkürzungen. Das erspart Schreibarbeit. Der besseren Lesbarkeit wegen verwende ich dennoch bei allen Beispielen die ausführliche Schreibweise.


```
Forward 80   bzw.    fd 80
Right 90             rt 90
Forward 80           fd 80
Right 90             rt 90
Forward 80           fd 80
```

und erhalten das nebenstehende Ergebnis:

`Pendown`

Mit dem Befehl `pendown` (abgekürzt `pd`) wird der Schreibstift abgesenkt und die Schildkröte zeichnet danach ihre Spur auf dem Bildschirm.

`Penup`

Mit dem Befehl `penup` (abgekürzt `pu`) wird der Schreibstift angehoben und die Schildkröte bewegt sich danach auf dem Bildschirm ohne eine Spur zu hinterlassen.

`ClearScreen`

Dieser Befehl (abgekürzt `cs`) löscht das Grafikfenster und setzt die Schidkröte auf die Ausgangsposition (0,0). Der Befehl ist äquivalent zur Kombination der Befehle `Clear` und `Home`.

Wir können nun die Schildkröte an eine andere Stelle bewegen und dort erneut ein Quadrat zeichnen lassen. Bei der Bewegung an den neuen Startpunkt soll sie keine Spur hinterlassen. Deshalb ist zunächst der Befehl

```
PenUp
```

einzugeben. Wird nun die Schildkröte mit z.B.

```
Forward 100
```

bewegt, muss anschließend der Befehl

```
PenDown
```

eingegeben werden, damit die nachfolgenden Bewegungen wieder eine Spur hinterlassen. Ein Quadrat, nun eventuell mit einer anderen Seitenlänge, kann dann durch erneute Eingabe der obigen Befehlsfolge gezeichnet werden. Die Befehle können auch in einer einzigen Zeile eingegeben werden, also z.B.

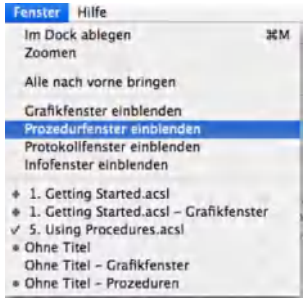
```
Forward 50 Right 90 Forward 50 Right 90 Forward 50 Right 90 Forward 50
```

Experimentieren Sie mit den Befehlen `Forward, Back, Left` und `Right`. Bewegen Sie die Schildkröte an verschiedene Orte und lassen Sie sie einfache Figuren zeichnen. Mit dem Befehl `ClearScreen` können Sie den Bildschirm komplett löschen und neue Grafiken zeichnen.

Erweiterbarkeit

Das mehrfache Zeichnen gleichartiger Figuren ist mühsam, denn bisher müssen Sie dazu die jeweilige Befehlsfolge immer wieder neu eingeben. Es gibt deshalb die Möglichkeit, solche Befehlsfolgen zusammenzufassen und als neuen Befehl mit einem bestimmten Namen zu belegen - wir sprechen dann von einer Prozedur. Der Befehlssatz von ACSLogo kann so um einen Befehl des gewählten Namens erweitert werden. Wir wollen also zunächst die Befehlsfolge zum Zeichnen eines Quadrats zusammenfassen und unter dem Namen "Quadrat" verfügbar machen.

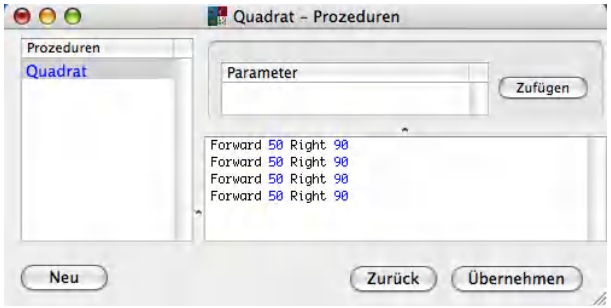
Befehlsfolgen werden in Prozeduren eingegeben. Dazu rufen Sie zunächst aus dem Menü `Fenster` die Option `Prozedurfenster einblenden` auf.



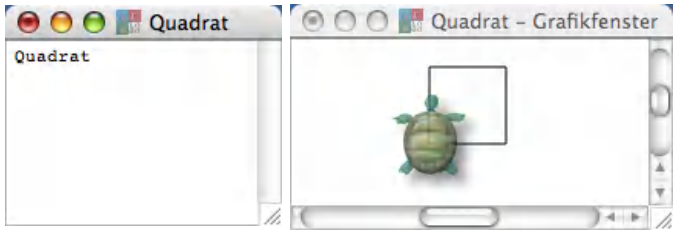
Das Prozedurfenster sieht folgendermaßen aus:

Links ist die Liste der von Ihnen bereits definierten Prozeduren (momentan ist diese Liste noch leer). Da wir eine neue Prozedur schreiben wollen, klicken Sie auf den <*Neu*> Schalter.

Ein Eintrag mit dem Namen `newproc1` wird hinzugefügt. Überschreiben Sie diesen mit dem Namen `Quadrat` und drücken Sie <*Return*>. Dann geben Sie die Befehlsfolge im Hauptbereich ein. Der letzte Befehl `Right 90` ist nicht für das Zeichnen des Quadrats notwendig. Es empfiehlt sich aber, ihn hinzuzufügen, da dann die Schildkröte nach Zeichnen des Quadrats in ihrer Ausgangsposition steht.



Beachten Sie, dass der Prozedurname blau ist, weil die Prozedur verändert (bzw. neu eingegeben) wurde und noch nicht angewendet wurde. Drücken Sie den Schalter <*Übernehmen*>, damit die Prozedur anwendbar wird. Wenn Sie jetzt im Eingabefenster den Befehl `Quadrat` eingeben und zur Ausführung bringen, wird im Grafikfenster das Quadrat gezeichnet.



Wiederholungen

`Repeat` *`Anzahl [Liste]`*

Die Definition von Prozeduren hat uns schon viel Schreibarbeit erspart. Wir können dies mit einer Kontrollstruktur wie dem `Repeat`-Befehl noch steigern, mit dem die Abfolge gleichartiger Befehlsfolgen zusammenfasst werden kann. Dies wiederholt die Befehlsfolge, die in der Liste aufgeführt ist, so oft, wie mit ***`Anzahl`*** angegeben. Die Liste muss immer von eckigen Klammern umschlossen sein (ALT-5 und ALT-6 auf der Tastatur)!

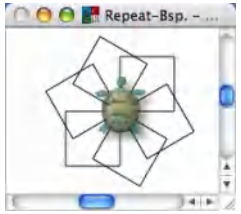
In unserem Beispiel tauchen die Befehle `Forward 50` bzw. `Right 90` jeweils viermal auf. Diese Wiederholung kann mit Hilfe des `Repeat`-Befehls einfacher ausgedrückt werden:

```
Repeat 4 [Forward 50 Right 90]
```

Sie können diese Prozedur aus der bereits vorhandenen Prozedur `Quadrat` erhalten, indem Sie im Prozedurfenster die überflüssigen Befehlszeilen löschen und dafür den Befehl `Repeat 4` einfügen. Geben Sie nun im Direktmodus den Befehl `Quadrat` ein, sollten Sie dasselbe Ergebnis erhalten wie mit der ersten Prozedur `Quadrat`.

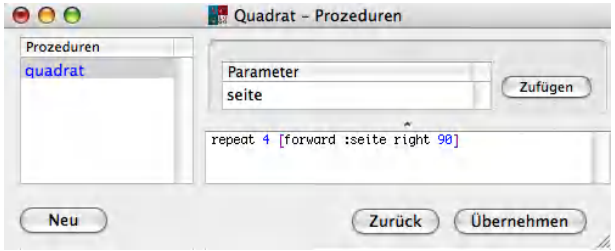
Besonders nützlich ist, dass dieser Befehl in sich selbst verwendet werden darf, d. h. dass wir verschachtelte Wiederholungs-Schleifen erstellen können:

```
Repeat 6 [Repeat 4 [Forward 50 Right 90]
Right 60]
```



Parameterübergabe

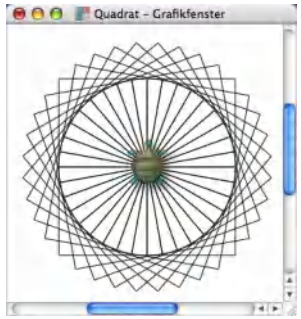
Das Quadrat-Beispiel wollen wir noch etwas weiter führen. Es sollen nun Quadrate mit jeweils unterschiedlichen Seitenlängen werden. Derzeit könnten wir das nur dadurch erreichen, dass wir innerhalb der Prozedur `Quadrat` die Seitenlänge jeweils neu eingeben und dann die so veränderte Prozedur aufrufen. Wesentlich eleganter und flexibler können wir dies allerdings lösen, wenn wir der Prozedur einen Wert für die Seitenlänge übergeben. Der Aufruf lautet dann z.B. `Quadrat 100` und die entsprechend veränderte Prozedur:



Der Parameter `:seite` ist also ein Platzhalter für den Zahlenwert, den wir für die Seitenlänge des Quadrats eingeben. Ein solcher Platzhalter ist automatisch für alle Prozeduren bekannt und gültig; es handelt sich um eine **globale** Variable. Manchmal macht es Sinn, dass Variablen nur innerhalb einer Prozedur bekannt und gültig sind (z. B. bei Zählvariablen), vor allem um unerwünschte Nebenwirkungen außerhalb der Prozedur zu vermeiden. Sie müssen dazu explizit mit dem Befehl `local` als **lokale** Variable deklariert werden. In den folgenden Kapiteln wird das von mir allerdings nur selten angewendet.

Der Befehl `Quadrat` (bzw. jeder andere Befehl aus selbstgeschriebenen Prozeduren) kann nun mit allen anderen Befehlen von ACSLogo kombiniert werden. So erhalten wir bei Eingabe der folgenden Befehlszeile die abgebildete Rosette:

```
repeat 36 [Quadrat 100 Right 10]
```



Wir haben in dieser Befehlszeile sichergestellt, dass mit 36 Drehungen um 10^0 gerade die Winkelsumme von 360^0 erfüllt ist.

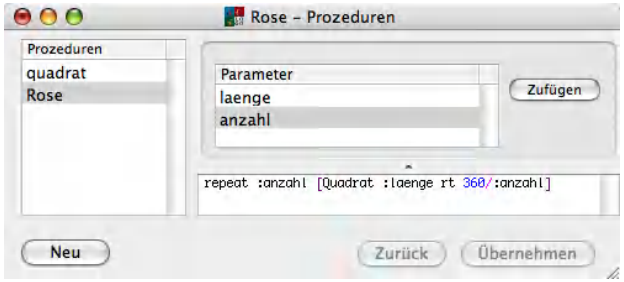
Durch die Verknüpfung von Parameter und Zahlen kann dies weiter vereinfacht und verallgemeinert werden. *Vereinfachung* und *Verallgemeinerung* ist ein durchgängiges Ziel bei den folgenden Beispielen, weil es ein systematisches Untersuchen der Eigenschaften eines Systems oder der Auswirkungen von Abbildungsvorschriften ungemein erleichtert.

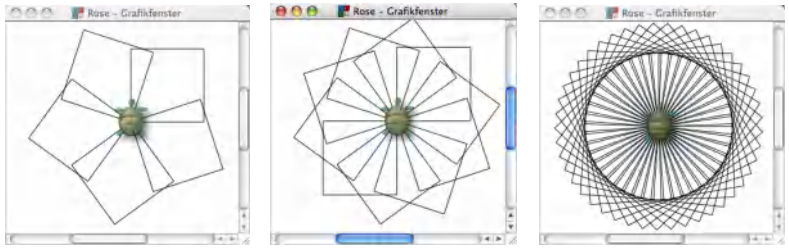
`Operationen mit Zahlen: + - * /`

Die Verknüpfungen verlangen zwei Zahlen als Eingaben und geben die Summe, die Differenz, das Produkt bzw. den Quotienten dieser Zahlen zurück.

Zur Ein- und Ausgabe von sehr großen Zahlen kann der Buchstabe e für „mal 10 hoch“ verwendet werden, d.h. `print 10e7` liefert den Wert `1e+08`.

In der folgenden Prozedur wird gleichzeitig deutlich, dass Prozeduren (hier eine Prozedur `Rose`) wiederum andere Prozeduren aufrufen können (hier die Prozedur `Quadrat`):





Prozedur Rose mit laenge 50 sowie anzahl 5 (links), 10 (Mitte) bzw. 45 (rechts)

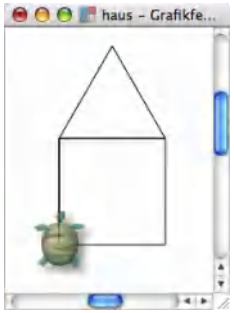
Zerlegung in Teilaufgaben

In den Beispielen des Quadrats und der Rose wurde gezeigt, wie einfach der Befehlssatz von ACSLogo erweitert werden kann. Umgekehrt erlaubt das Prozedurkonzept die Zerlegung komplexer Aufgaben in einfachere Teilaufgaben.

Soll z.B. eine Grafik erzeugt werden, die ein Haus darstellt wie in der Abbildung, so können wir die Aufgabe zerlegen in die Teilaufgabe des Zeichnens von Hausbestandteilen wie Viereck als Hausfront und Dreieck als Dachstock[7]. Das Gesamtprogramm kann dann folgendes Aussehen haben (die Prozedur `Quadrat` braucht nicht mit eingegeben zu werden, sofern sie sich noch im Editor befindet):

[7] Die Programmierung eines Hauses findet sich bei Papert (1982, S. 38 f.) als prominentes Beispiel für das Lernen aus Fehlern.

Das ergibt die Prozedur **Haus**:




```
Quadrat :seite
forward :seite
Dreieck :seite
back :seite
```

Aufgerufen wird u.a. die Prozedur **Dreieck**:

```
right 30
repeat 3 [forward :seite right 120]
left 30
```

Die Prozedur **Dreieck** soll ein gleichseitiges Dreieck zeichnen, d.h. bei einer Winkelsumme von 180^0 soll jeder Innenwinkel 60^0 betragen. Es ist deshalb zu beachten, dass mit dem Befehl **right** derjenige Außenwinkel übergeben wird, der zusammen mit dem gewünschten Innenwinkel 180^0 ergibt. Diese möglicherweise überraschende Notwendigkeit ergibt sich aus der Tatsache, dass die Schildkröte alle Bewegungen und Änderungen der Blickrichtung aus ihrer aktuellen Position heraus vollzieht, sich also nicht innerhalb eines kartesischen Koordinatensystems bewegt (vgl. dazu S. 57).

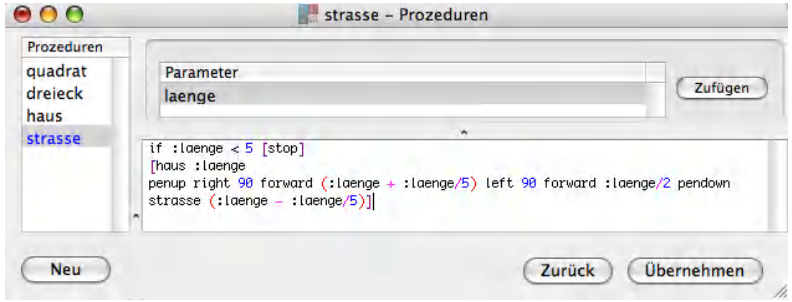
In der Prozedur **Haus** wird darauf geachtet, die Schildkröte nach Ausführung des Befehls (also Zeichnen des Quadrats bzw. des Dreiecks) am Ausgangspunkt mit ursprünglicher Blickrichtung zu hinterlassen.

Rekursion

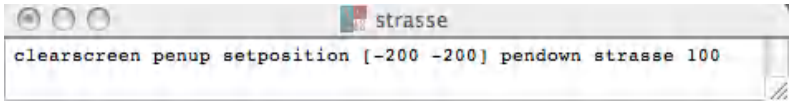
Nachdem das Zeichnen eines Hauses mit einem Befehl möglich ist, soll als nächstes ein ganzer Straßenzug dargestellt werden. Dazu ist die Schildkröte am besten zuerst (mit "gehobenem Stift") in die linke untere Bildschirmecke zu bewegen.

Die Prozedur besteht im wesentlichen darin, dass ein Haus gezeichnet wird, anschließend die Schildkröte etwas nach rechts und oben versetzt wird, und dann erneut ein Haus gezeichnet wird. Das erneute Zeichnen eines Hauses erfolgt dadurch, dass sich die Prozedur **Strasse** selbst (rekursiv) mit reduzierter Seitenlänge aufruft. Dieser Selbstaufruf am Ende einer Prozedur wird auch als *endständige Rekursion* bezeichnet (vgl. Kap. Vielecke und Spiralen, S. 25 f.).

Allgemein gesprochen ist die Rekursion der Selbstaufruf einer wiederholt abzuarbeitenden Folge von Anweisungen. Typischerweise ist eine Abbruchbedingung zu formulieren, nach deren Eintreten der Selbstaufruf beendet wird. Da in unserem Beispiel die Seitenlänge bei jedem Aufruf von **Strasse** verkürzt wird, ist am Anfang der Prozedur sichergestellt, dass bei Unterschreiten einer bestimmten Seitenlänge der ganze Vorgang beendet wird. Sofern die Abbruchbedingung noch nicht erfüllt ist, wird das Haus gezeichnet. Wird auf eine solche Abbruchbedingung verzichtet, läuft die entsprechende Aktion unkontrolliert weiter.



Der Aufruf der Prozedur (mit vorheriger Neupositionierung der Schildkröte) kann dann wie folgt aussehen:



If *Bedingung [true-Befehlsfolge] [false-Befehlsfolge]*

Mit dem Schlüsselwort **If** wird die Kontrollstruktur der bedingten Anweisung beschrieben. Wenn die *Bedingung* zutrifft, wird die erste Befehlsfolge ausgeführt, ansonsten wird die zweite Befehlsfolge ausgeführt. Beide Befehlslisten müssen vorhanden sein, auch wenn eine der Listen leer ist, also keine Befehlsausführung nach sich zieht.

And *Bedingung_1 Bedingung_2*
(And *Bedingung_1 Bedingung_2 ... Bedingung_n***)**

Mit dem logischen Operator UND wird eine sowohl-als-auch Verknüpfung formuliert. Wenn also beide (alle) Bedingungen wahr sind, wird der Wert wahr (true) geliefert, wenn eine Bedingung nicht zutrifft, der Wert falsch (false). Zu beachten ist, dass bei mehr als zwei Bedingungen die gesamte Abfrage mit Klammern einzufassen ist.

Or *Bedingung_1 Bedingung_2*
(Or *Bedingung_1 Bedingung_2 ... Bedingung_n***)**

Mit dem logischen Operator ODER wird eine entweder-oder Verknüpfung formuliert. Wenn also mindestens eine Bedingung wahr ist, wird der Wert wahr (true) geliefert, nur wenn beide (alle) Bedingungen nicht zutreffen, der Wert falsch (false).

Stop

Mit **Stop** wird die Abarbeitung einer Prozedur beendet und ohne Übergabe eines Wertes an die die Prozedur aufrufende Stelle zurück gekehrt. Üblicherweise wird dieser Befehl im Zusammenhang mit der **If**-Anweisung zur Beendigung eines Rekursionsaufrufs verwendet.

SetPosition *[x y]*

Mit **SetPosition** wird die Schildkröte zur Position **x,y** bewegt. Ist der Schreibstift abgesenkt, wird eine Linie in der aktuellen Farbe gezeichnet.

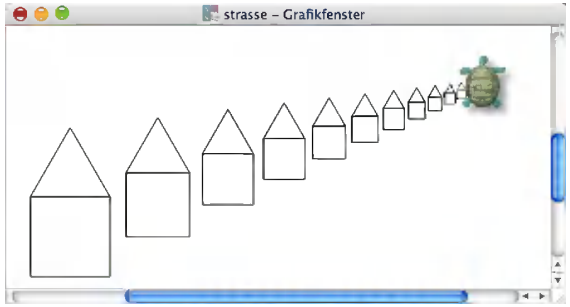
SetX *x*

Mit **SetX** wird die Schildkröte zur Position mit der X-Koordinate **x** bewegt unter Beibehaltung der Y-Koordinate.

SetY *y*

Mit **SetY** wird die Schildkröte zur Position mit der Y-Koordinate **y** bewegt unter Beibehaltung der X-Koordinate.

Die Prozedur **Strasse** liefert dann folgendes Ergebnis:



Inzwischen kennen Sie einige Befehle von ACSLogo und die wichtigsten Funktionen der Entwicklungsumgebung. In den weiteren Kapiteln werden diese Kenntnisse angewandt und vertieft. Einige der Anwendungen sind Ihnen wahrscheinlich schon bekannt, andere sind für Sie vielleicht neu und regen an zu eigenen Experimenten.

Vielecke und Spiralen

Als Einführungsbeispiel hatten wir das Zeichnen eines Quadrats kennen gelernt. Die dort erstellte Prozedur soll Ausgangspunkt für einige Verallgemeinerungen werden. Die in dem Beispiel erreichte Wiederholung über den **repeat**-Befehl können wir auch über einen rekursiven Aufruf der Prozedur erreichen (damit die Prozedur nicht endlos läuft, prüfen wir zuvor noch, ob die Schildkröte den Ausgangspunkt erreicht hat):

```
Prozedur quadrat seite
forward :seite right 90
if (and (abs xpos) < 0.1 (abs ypos) < 0.1) [stop] [quadrat :seite]
```

Wir können daraus leicht eine Prozedur für Vielecke (n-Ecke) machen, in der ein Parameter **n** die Zahl der Ecken des Vielecks bestimmt. Dies stellt eine Verallgemeinerung dar, die eine Vielzahl daraus abgeleiteter Figuren erlaubt:

```
Prozedur n_eck n seite
forward :seite right 360/:n
if (and (abs xpos) < 0.1 (abs ypos) < 0.1) [stop] [n_eck :n :seite]
```

Die Abbildung zeigt mit der Prozedur **n_eck** erzeugte Figuren (von n=3 bis n=7).

Versuchen Sie doch, mit der Prozedur ***n_eck*** *einen Kreis zu erzeugen. Wie lassen sich Kreise unterschiedlicher Radien erzeugen?*

Abs *Zahl*

Abs liefert den Absolutwert der ***Zahl.*** Bei weiterer Verrechnung der Werte beachten Sie die Klammerregel, bei der gilt: **(abs -x) + (abs -y)**

XPos

XPos liefert die X-Koordinate der aktuellen Position der Schildkröte.

Ypos

YPos liefert die X-Koordinate der aktuellen Position der Schildkröte.

Als nächsten Schritt lassen wir die Seitenlängen unserer Vielecke zunehmen (oder abnehmen) - festgelegt durch einen Parameter **zunahme** - und erhalten daraus spiralige Figuren (den Drehwinkel legen wir ebenfalls über einen Parameter **winkel** fest):

```
Prozedur spirale seite winkel zunahme
if :seite < 150
[forward :seite right :winkel
 Spirale :seite + :zunahme :winkel :zunahme]
```

spirale 5 120 5

spirale 5 90 2

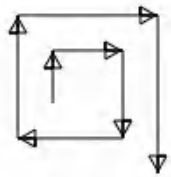
spirale 5 144 5

spirale 2 121 4

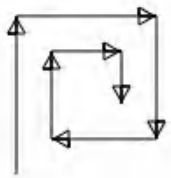
An den Spiralen lässt sich der rekursive Ablauf gut nachvollziehen, was für deren Verständnis sehr hilfreich ist. In der Prozedur `spirale` wird ja zuerst eine Strecke mit der Länge `:seite` gezeichnet und anschließend erneut die Prozedur `spirale` aufgerufen (endständige Rekursion), nun mit der verlängerten Strecke `:seite+:zunahme`.

In einer modifizierten Prozedur `spiraleaR` rufen wir nun zuerst erneut die Prozedur `spiraleaR` auf (anfangsständige Rekursion) - also bevor die Strecke gezeichnet wird - und zwar solange, bis das Abbruchkriterium erfüllt ist. Hier wird erst im „Rücklauf", also jeweils nach Rücksprung zur rufenden Stelle die Streckenzeichnung mit der bis dort aufsummierten Länge vollzogen.

```
Prozedur spiraleaR seite winkel zunahme
if :seite < 150
[SpiraleaR :seite + :zunahme :winkel :zunahme
 forward :seite right :winkel]
```



spirale 30 90 10



spiraleaR 30 90 10

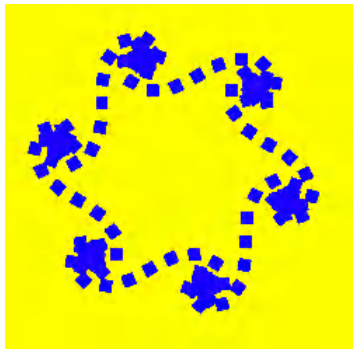
Die Unterscheidung der Rekursionstypen erfolgt also nach dem Ort des Selbstaufrufs.:

anfangsständige Rekursion	**endständige** Rekursion
• Abbruchbedingung • Selbstaufruf der Prozedur • Anweisung(en)	• Abbruchbedingung • Anweisung(en) • Selbstaufruf der Prozedur

Die unterschiedliche Abarbeitung der Befehlsfolgen lässt sich übrigens in ACSLogo im Einzelnen gut im `Protokollfenster` verfolgen. Mit diesen Rüstzeug können nun weitere interessante Spiralformen entwickelt werden.

Zunächst können wir eine ganz leicht abgewandelte Prozedur untersuchen, die ich `nanu` genannt habe, weil sie doch überraschende Formen ergibt. Es ist keine Abbruchbedingung formuliert; der Programmablauf muss deshalb mit (⌘+ .) gestoppt werden.

```
Prozedur nanu seite winkel zunahme
forward :seite right :winkel
nanu :seite :winkel + :zunahme :zunahme
```



Testen Sie die Prozedur z.B. mit den Aufrufen

```
nanu 10 10 5
nanu 10 10 12
nanu 10 40 7
nanu 10 40 30
nanu 10 2 20
```

Wenn Sie das Zeichnen einer Linie ersetzen durch einen Sprung ans Linienende und dort ein Polygon zeichnen, z.B. ein Quadrat, oder einen dicken Strich, dann erhalten Sie Figuren wie in der nebenstehenden Abbildung.

Mit den Befehlen `SetBackground`, `SetPenColor` und `SetPenWidth` können Sie die Darstellungen verändern und abwechslungsreich gestalten.

`SetBackground` ***Farbcode***

`SetBackground` (abgekürzt `SetBG`) setzt die Hintergrundfarbe im Grafikfenster gemäß dem Farbcode. Wird erst nach Aufruf des Befehls `ClearScreen` oder `Clean` wirksam.

`SetPenColor` ***Farbcode***

`SetPenColor` (abgekürzt `SetPC`) setzt die Stiftfarbe gemäß dem Farbcode. Wirkt für Linien und Flächenfüllungen.

`SetPenWidth` ***Breite***

`SetPenWidth` setzt die Linienbreite gemäß dem Wert ***`Breite`***.

Die 16 verfügbaren Farben sind über folgenden Farbcode ansprechbar:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Damit können wir auch unsere Spiralen in neuer Farbenpracht erstellen. Dazu setzen wir zunächst an alle gezeichneten Linien der Spirale ein Quadrat an. Die Prozedur `quadrat` wird nun ihrerseits so erweitert, dass das gezeichnete Quadrat mit einer zufällig aus den 15 Farben gewählten Farbe gefüllt wird (Weiß mit Farbcode 0 wird davon ausgeschlossen). Die zugehörigen Prozeduren sehen dann aus wie folgt:

```
Prozedur spirale seite winkel zunahme
```

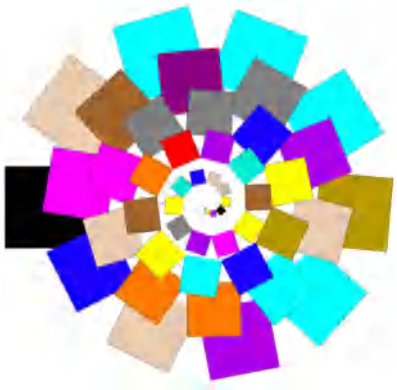
```
if :seite < 150
[forward :seite right :winkel
          quadrat :seite
Spirale :seite + :zunahme :winkel :zunahme]
```

```
Prozedur quadrat seite
```

```
setpc 1 left 90
repeat 4[forward :seite right 90]
right 90
setpc 1 + random 15
make "weg currentpath fillpath :weg
penup pendown
```

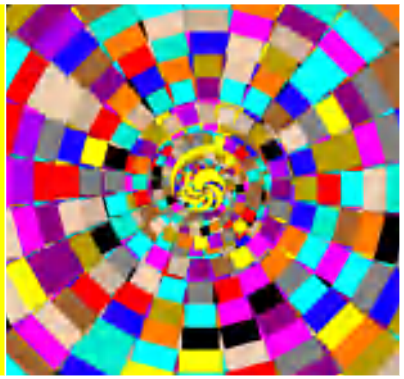
Der Aufruf von `spirale` bzw. `spiraleaR` liefert dann z.B. Bilder wie diese:

spirale 1 37 3



spiraleaR 1 37 3

sternspirale 7



wirbel 6

Das Füllen von Polygonen mit Farbe ist leider in ACSLogo mit den in anderen Logo-Versionen üblichen Befehlen wie `Fill` oder `FillIn` nicht verlässlich möglich. Es wird aber eine zwar leicht umständlichere jedoch durchaus Logo-adäquate Erweiterung bereit gestellt mit `FillPath`, womit eine Liste der gezeichneten Linien zur Verfügung steht. Anhand dieser Liste können dann vergleichbare Funktionen ausgeführt werden.

`Random` *`Zahl`*

Bei Eingabe einer positiven ganzen *`Zahl`* wird eine Zufallszahl aus dem Intervall zwischen 0 und `(Zahl - 1)` zurück gegeben.

`Make` *`Name Wert`*

Weist der Variablen *`Name`* den *`Wert`* zu. Wenn die Variable noch nicht existiert, wird sie neu definiert. Der *`Name`* muss immer ein Wort sein, als *`Weret`* sind Zahlen, Texte oder Listen zulässig.

`CurrentPath`

Mit `CurrentPath` wird eine Liste geliefert, die die Abfolge der durch die Bewegungen der Schildkröte gezogenen Linien enthält. Diese Liste kann als Eingabe für den Befehl `FillPath` genutzt werden.

`FillPath` *`[Path-Liste]`*

Mit `FillPath` wird eine Fläche farbig gefüllt, die von den durch die `Path-Liste` gelieferten Linien umschlossen wird. Wichtig: Die `Path-Liste` kann durch die Befehle `PenUp` und `PenDown` geleert werden.

`SetHeading` *`Zahl`*

Die Schildkröte dreht sich in die durch *`Zahl`* angegebene Richtung. Die *`Zahl`* wird als Gradangabe interpretiert, d.h. 0° zeigt nach oben, 180° nach unten. Die Gradangaben nehmen in Uhrzeigerrichtung zu.

Die Beispiele **sternspirale** und **wirbel** verdeutlichen dies. Die Prozedur **sternspirale** ist eine nichtrekursive Version der Spirale, die Prozedur **wirbel** ist ein Mehrfachaufruf der Prozedur **spirale**.

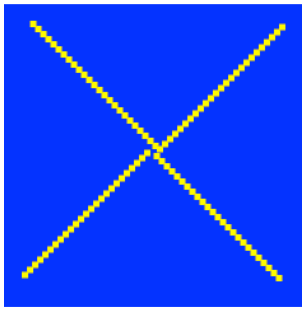
In ACSLogo fehlt der Befehl **for** *für das Abarbeiten einer Schleife. Die hier verwendeten Prozeduren sind Beispiele dafür, wie dieser Befehl durch den* **repeat***-Befehl in Kombination mit einer Zählvariable, die mit dem* **make***-Befehl hochgezählt wird, ersetzt werden kann. Diese Konstruktion wird an vielen Stellen notwendig werden.*

```
Prozedur sternspirale n
setpc 1 make "i 0
repeat 30
    [repeat :n [forward :i right 1080/:n]
     forward :i right 30 make "i :i + 4
     setpc 1 + random 15 make "weg currentpath fillpath :weg penup
     pendown]
```

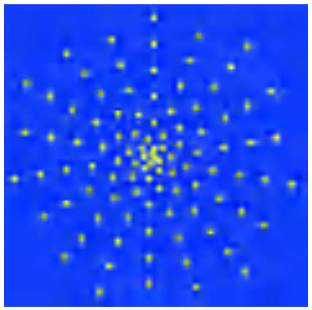
```
Prozedur wirbel n
make "i 0
repeat :n
    [pendown Spirale 2 15 0.5 penup
     setpos [0 0] setheading :i*360/:n make "i :i +1]
```

Hatten wir weiter oben das Zeichnen der Spiralen verkompliziert, indem an den Linien Quadrate angefügt wurden, soll es hier wieder vereinfacht werden, indem nur noch die Linienenden als Punkte gezeichnet werden. Die Variable **zunahme** wirkt dabei wie ein Vergrößerungsfaktor bzw. wie eine Lupe.

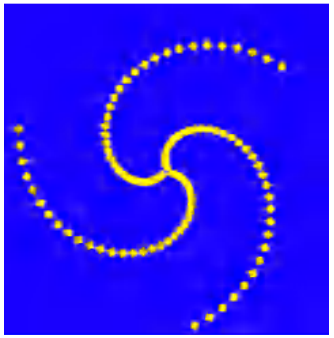
```
Prozedur punktspirale seite winkel zunahme
if :seite > 200 [stop]
     [fd 5 back 5 right :winkel
      penup forward :seite pendown
      punktspirale :seite + :zunahme :winkel :zunahme]
```



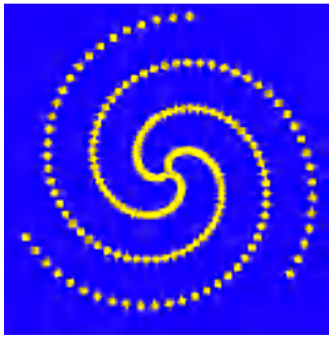
punktspirale 1 90 2



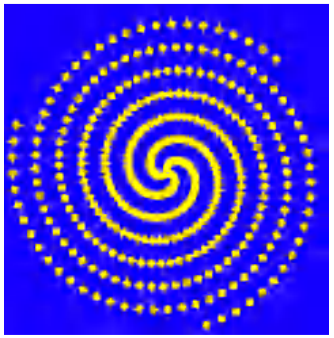
punktspirale 1 100 2



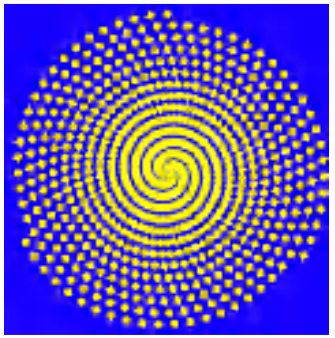
punktspirale 1 122 2



punktspirale 1 122 1



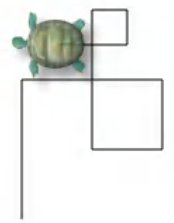
punktspirale 1 122 0.5



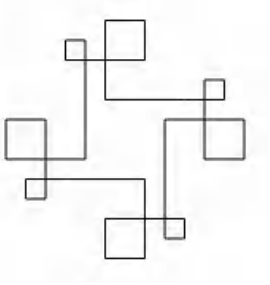
punktspirale 1 122 0.25

Das Kapitel über Vielecke wäre unvollständig ohne Vorstellung einer Prozedur, die in mehreren Logo-Büchern aufgegriffen und variiert wurde. Vermutlich geht sie auf Abelson (1983) zurück. Diese Prozedur, die als **dasDing** bezeichnet wird, zeichnet zunächst nur ein abstraktes Muster. Der mehrfache bzw. der rekursive Aufruf zeigt dann überraschend symmetrische Muster.

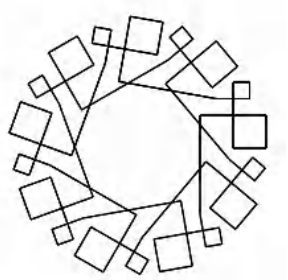
```
Prozedur dasDing seite
forward :seite right 90
forward :seite right 90
forward :seite/2 right 90
forward :seite/2 right 90
forward :seite right 90
forward :seite/4 right 90
forward :seite/4 right 90
forward :seite/2
```



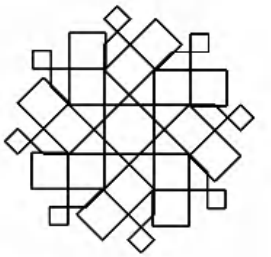
Der Aufruf der Prozedur in Wiederholungsschleifen und zusätzlichen Drehungen liefert dann z.B. folgende Gebilde:



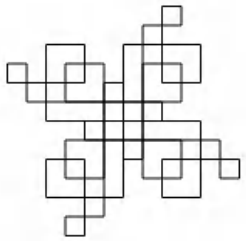
repeat 4 [dasDing :seite]



repeat 9 [dasDing :seite right 10 forward :seite/2]



repeat 9 [dasDing :seite left 45 forward :seite]

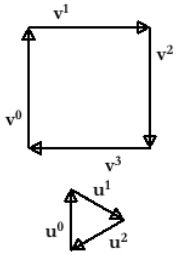


repeat 4 [dasDing :seite dasDing :seite left 90]

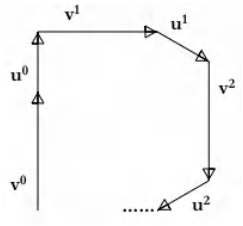
Mit den hier entwickelten Prozeduren lässt sich bereits ausgiebig experimentieren. Nun sollte sich auszahlen, dass die Prozeduren immer sehr flexibel angelegt sind, d.h. durch die Verwendung von Übergabeparametern einfach und vielfältig änderbar sind.

Wie auf den letzten Seiten gezeigt, lassen sich bei geeigneter Wahl der Hintergrundfarbe sowie Farbe und Dicke des Zeichenstifts schöne grafische Effekte erzielen.

Ich möchte dieses Kapitel abschließen mit der Vektordarstellung von Polygonen (Näheres dazu bei Abelson & DiSessa, 1981, S. 105 ff.), die uns weitere Flexibilität liefert. Ausgangspunkt bei dieser Überlegung sind zwei Polygone, die nun aber miteinander kombiniert werden, wie in der folgenden Abbildung dargestellt: Jedes Polygon wird als Abfolge von Vektoren betrachtet, die jeweils durch Ausgangspunkt, Länge und Richtung der zu zeichnenden Strecke gekennzeichnet sind. Hier gilt also nicht mehr die Turtlegrafik, bei der immer von der aktuellen Blickrichtung ausgegangen wird, sondern der absolute Winkel. Entsprechend formulieren wir eine Prozedur `vektor` (entgegen den mathematischen Konventionen bedeutet hier eine Ausrichtung 0 die Ausrichtung nach oben, nicht nach rechts. Dies ist für das weitere Vorgehen aber belanglos).



Vektorendarstellung eines Quadrats und eines Dreiecks



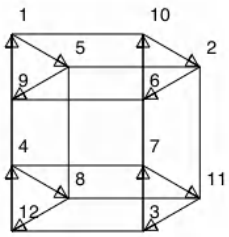
Vektorendarstellung der Kombination eines Quadrats mit einem Dreieck

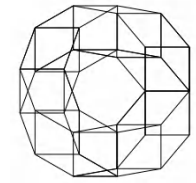
```
Prozedur vektor richtung laenge
setheading :richtung
forward :laenge
```

Zur Kombination zweier Polygone dient dann eine Prozedur `zweipolygone`, in der jeweils im Wechsel die Prozedur `vektor` mit den Parametern für das erste bzw. zweite Polygon aufgerufen wird.

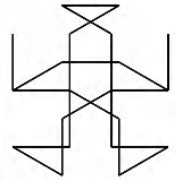
```
Prozedur zweipolygone seite1 winkel1 seite2 winkel2 n
make "z 0
repeat :n [vektor :z*:winkel1 :seite1
           vektor :z*:winkel2 :seite2
           make "z :z+1]
```

Es lohnt sich, am einfachen Beispiel alle Schritte nachzuvollziehen, die von dieser Prozedur abgearbeitet werden, bevor kompliziertere Kombinationen ausprobiert werden, wie nebenstehend die oben besprochene Kombination eines Quadrats und eines Dreiecks: `zweipolygone 100 90 50 120 12`

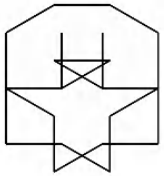




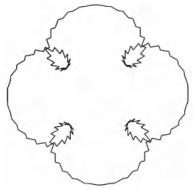
zweipolygone 50 90 50 320 50



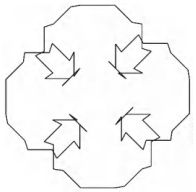
zweipolygone 50 300 50 90 50



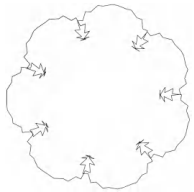
zweipolygone 50 60 50 90 50



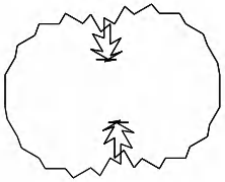
zweipolygone 15 20 8 4 200



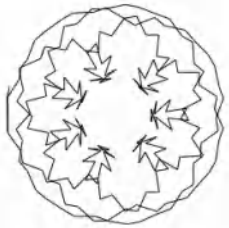
zweipolygone 50 45 20 9 200



zweipolygone 35 32 20 4 200



zweipolygone 20 30 12 10 100



zweipolygone 20 33 12 10 110

zweipolygone 20 10 20 -15 250

Interessanterweise sind es geschlossene Figuren, die die Prozedur liefert. Nach Abelson & DiSessa (1981, S. 118) lässt sich der Symmetriewinkel der Figuren errechnen: Teilen wir 360° durch (winkel1/winkel2 - 1) liefert dies den Symmetriewinkel, bei 90° also Vierfach-Symmetrie, bei 72° Fünffach-Symmetrie usw.

Nun soll in einem weiteren Schritt die Prozedur **zweipolygone** so verallgemeinert werden, dass beliebig viele Polygone kombiniert werden können. Umgesetzt wird dies mit einer Prozedur **mehrpolygone**, in der wir intensiv mit Listen arbeiten.

Prozedur **mehrpolygone** *paare n*

```
make "m count :paare
if :m = 0 [stop][make "z 0
   repeat :n [make "paarliste :paare
   repeat :m [make "paar first :paarliste make "test count :paar
          if :test = 0 [stop]
          [make "s first :paar make "w last :paar
          vektor :z*:w :s make "paarliste butfirst :paarliste]]
           make "z :z+1]]
```


Word *Wort_1 Wort_2*
(Word *Wort_1 Wort_2 ... Wort_n)*

Aus einer variablen Anzahl von Wörtern als Eingaben (Voreinstellung zwei) wird ein Wort als Verkettung gebildet und zurück gegeben.

List *Objekt_1 Objekt_2*
(List *Objekt_1 Objekt_2 ... Objekt_n)*

Aus einer variablen Anzahl von Eingaben (Voreinstellung zwei) wird eine Liste aus diesen Eingaben gebildet und zurück gegeben. Die Objekte können Texte, Zahlen oder wiederum Listen sein.

Count *Wort;* **Count** *Liste*

Mit **Count** wird die Anzahl der Elemente eines Objekts zurück gegeben, also im Falle eines Wortes die Anzahl der Zeichen, im Falle einer Liste die Zahl der Listenelemente.

Empty? *Wort;* **Empty?** *Liste*

Mit **Empty?** (alternativ **EmptyP**) wird geprüft, ob ein Objekt keine Elemente besitzt. **Empty?** liefert also true im Falle eines Wortes, wenn die Anzahl der Zeichen Null ist, im Falle einer Liste, wenn es sich um eine leere Liste handelt.

First *Wort;* **First** *Liste*

Bei Eingabe eines Wortes wird dessen erster Buchstabe, bei einer Liste deren erstes Element zurück gegeben. Bei Eingabe eines leeren Wortes oder einer leeren Liste erfolgt eine Fehlermeldung.

Last *Wort;* **Last** *Liste*

Bei Eingabe eines Wortes wird dessen letzter Buchstabe, bei einer Liste deren letztes Element zurück gegeben. Bei Eingabe eines leeren Wortes oder einer leeren Liste erfolgt eine Fehlermeldung.

ButFirst *Wort;* **ButFirst** *Liste*

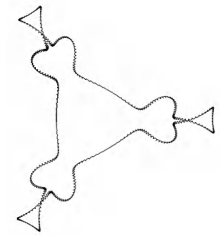
Bei Eingabe eines Wortes wird dieses Wort ohne dessen ersten Buchstaben, bei einer Liste diese Liste ohne deren erstes Element zurück gegeben. Bei Eingabe eines leeren Wortes oder einer leeren Liste erfolgt eine Fehlermeldung.

ButLast *Wort;* **ButLast** *Liste*

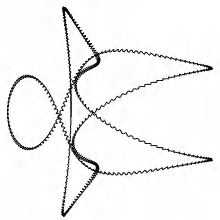
Bei Eingabe eines Wortes wird dieses Wort ohne dessen letzten Buchstaben, bei einer Liste diese Liste ohne deren letztes Element zurück gegeben. Bei Eingabe eines leeren Wortes oder einer leeren Liste erfolgt eine Fehlermeldung.

In dieser Prozedur wird für jedes Polygon ein Wertepaar für die Seitenlänge und den Winkel festgelegt und die Gesamtheit dieser Wertepaare in einer Liste zusammen gefasst. So enthält zB. **mehrpolygone [[2 1] [2 -8] [2 16] [2 -2]]** vier Polygone. Das Prinzip der Prozedur besteht darin, dass jeweils die erste Seite des ersten Polygons gezeichnet wird, dann die erste Seite des zweiten Polygons usw. bis die Liste abgearbeitet ist durch Mehrfachaufruf der Prozedur **vektor**. Dann wird wieder mit dem ersten Polygon der Liste begonnen, d.h. mit dessen zweiter Seite, und diese Abfolge dauernd fortgesetzt. Hiermit lassen sich unüberschaubar viele interessante und häufig skurrile Figuren generieren!

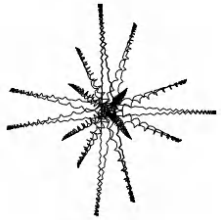
Da die Prozedur **`mehrpolygone`** *deutlich anders funktioniert als die Prozedur* **`zweipolygone`** *sollten Sie auch hier bekannte Wertekombinationen verwenden und daran den internen Ablauf der Prozedur nachvollziehen und kontrollieren, also z.B.* **`mehrpolygone [[50 90] [50 320]] 50`**. *Das sollte dann dieselbe Figur ergeben wie* **`zweipolygone 50 90 50 320 50.`**



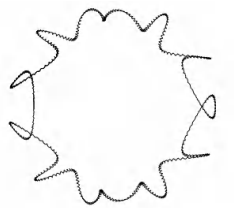
mehrpolygone [[2 1] [2 -8] [2 16] [2 -2]] 400



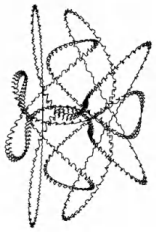
mehrpolygone [[2 2] [2 -3] [2 5] [2 -7]] 400



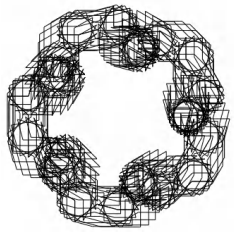
mehrpolygone [[4 7] [4 -8] [4 9] [4 -10]] 400



mehrpolygone [[2 1] [2 -12] [2 16] [2 -10]] 400

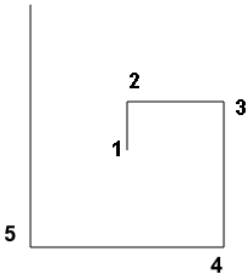


mehrpolygone [[4 8] [4 -13] [4 14] [4 -12] [4 3] [4 -8] [4 9] [4 -8]] 500



mehrpolygone [[30 90] [30 15] [40 88]] 400

Spirolaterale



Von den Spiralen zu den Spirolateralen! Diese interessanten grafischen Gebilde sind in ihrer Grundform sehr simpel zu konstruieren: Begonnen wird mit einer Linie bestimmter Länge, gefolgt von einer Rechtsdrehung um 90^0. Daran angesetzt wird die nächste Linie mit doppelter Länge, es folgt erneut eine 90^0-Drehung, dann eine Linie mit dreifacher Länge usw. Dies wird bis zu einer maximal vorgegebenen Anzahl von Iterationen fortgesetzt (im rechten Beispiel also fünf mal). Diese Schleife kann nun mehrfach wiederholt werden. Der ganze Vorgang wird abgebrochen, wenn die Schildkröte ihren Ursprungsort erreicht hat.

Spirolaterale wurden zuerst beschrieben von dem britischen Biochemiker Odds (1973)[8]. Zum Hintergrund und zur Umsetzung in Logo finden sich Anregungen bei Abelson & diSessa (1981, S. 37 ff.), Fisher & Campbell (1991) sowie bei Krawczyk (1999, 2000). Die dort gezeigten Beispiele können mit den hier entwickelten Prozeduren unmittelbar nachvollzogen werden.

Wir definieren dazu eine Prozedur **`spirolaterale`**. Aufgerufen wird dort die Prozedur **`spiro`** (in der die Linien gezeichnet werden) mit Übergabe der Variable `n`, die angibt, wie oft Linien und Drehungen vollzogen werden sollen. Danach wird geprüft, ob die Schildkröte ihren Ausgangspunkt erreicht hat. Falls ja, wird die Prozedur beendet, falls nein, wird **`spirolaterale`** erneut ausgeführt (endständige Rekursion).

Prozedur **`spirolaterale`** *`n`*

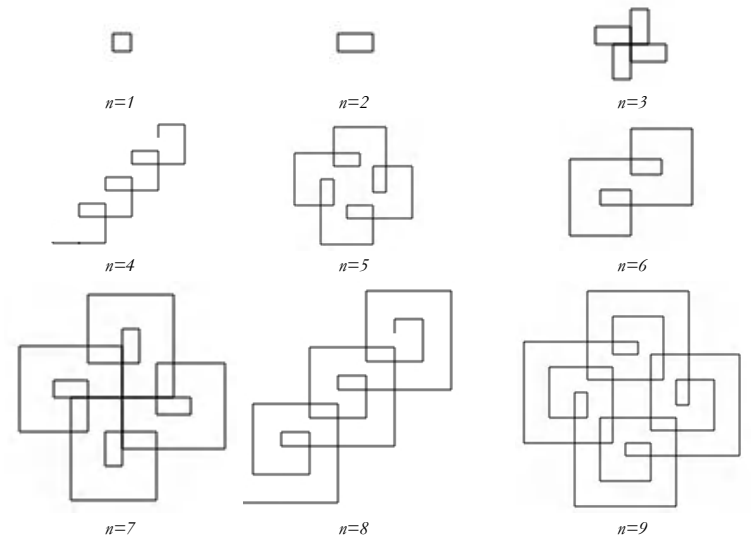
```
spiro :n
if (and (abs xpos) < 1 (abs ypos) < 1) [stop]
[spirolaterale :n]
```

Prozedur **`spiro`** *`n`*

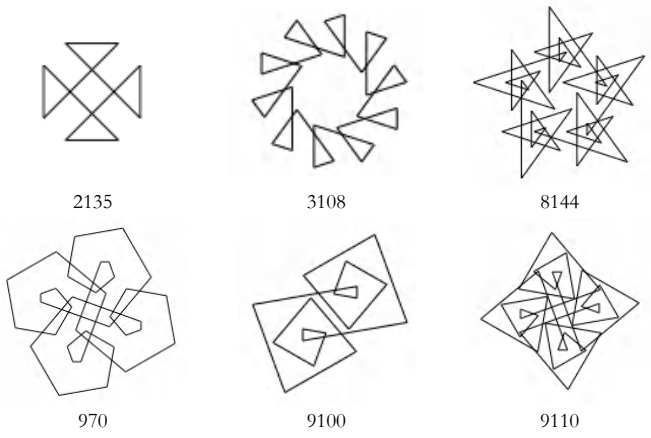
```
make "zaehler 1
repeat :n [forward 10*:zaehler right 90 make "zaehler :zaehler+1]
```

Als Drehungswinkel sehen wir hier also konstant 90^0 vor, die Linienlänge beträgt ein n-faches von 10 Bildpunkten (später werden wir das variabel gestalten). Der Aufruf **`spirolaterale n`** (mit n=1,2,...,9) führt dann zu den in der nächsten Abbildung gezeigten Ergebnissen. Dabei ist zu bemerken, dass für n=4 und mehrfachen davon keine geschlossenen Figuren erhalten werden (der Programmablauf muss deshalb dann mit (⌘+ .) gestoppt werden). Warum das so ist, wird bei Krawczyk (1999) gezeigt.

[8] In einem Brief an Krawczyk schildert Odds (2003), der heute ein renommierter Mykologe (Pilzforscher) ist, sehr anschaulich wie er spielerisch die Spirolateralen entwickelte und erst von Gardner (Scientific American) dazu animiert wurde, überhaupt darüber zu publizieren.



Wenn nun die Spirolateralen systematisch oder spielerisch untersucht werden sollen, macht es Sinn, die Linienlänge und den Drehungswinkel variabel zu gestalten (die so leicht veränderten Prozeduren brauchen hier nicht abgedruckt werden). Odds (1973) hat dafür folgende Notation eingeführt: n_w wobei n die Anzahl der Linien bzw. Drehungen ist und w der Drehungswinkel. Beispiele dafür sind in der folgenden Abbildung gezeigt.



Bis hierher konnten wir sehen, was passiert, wenn andere als rechte Winkel genommen werden. Aber es kann noch mehr variabel gestaltet werden. Was passiert, wenn nicht nur Rechtsdrehungen, sondern auch Linksdrehungen erlaubt sind? Und was passiert, wenn die Linien nicht konstant wachsen, sondern ihre Länge für jeden Schritt frei festgelegt werden kann? Bei der Suche nach Antworten auf diese Fragen kommen nun wieder Wörter und insbesondere die Listen ins Spiel, die Logo von der Programmiersprache LISP geerbt hat (LISP steht ja gerade für LISt Processing).

Besonders einsichtig ist wohl, wenn wir für eine Rechtsdrehung r eingeben können und für eine Linksdrehung l. Ein vierfacher Wechsel von Rechts- und Linksdrehungen könnte dann einfach als das Wort `rlrlrlrl` eingegeben werden[9]. Der Prozedur **`spirolaterale`** sind nun die Werte für Anzahl, Linienlänge, Drehungswinkel sowie das Wort mit den Richtungswechseln zu übergeben, womit diese ihrerseits die Prozedur **`spiro`** aufruft. Die Prozeduren bekommen nun folgendes Aussehen:

Prozedur **spirolaterale** *n laenge winkel richtungen*

```
spiro :n :laenge : winkel :richtungen
if  (and (abs xpos) < 1 (abs ypos) < 1) [stop]
[spirolaterale :n :laenge : winkel :richtungen]
```

Prozedur **spiro** *n laenge winkel richtungen*

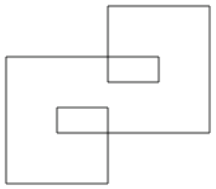
```
make "zaehler 1
repeat :n
 [forward :laenge*:zaehler
  make "richtung first :richtungen
      if :richtung = "r [right :winkel] [left :winkel]
 make "zaehler :zaehler+1  make "richtungen butfirst :richtungen]
```

Die Linienlänge kann nun auch noch variabel festgelegt werden, indem wir in einer Liste **`laengen`** gemäß der Linienabfolge nach jeder Drehung den jeweiligen Verlängerungsfaktor der Linie angeben. In der zweiten Liste **`richtungen`** legen wir dann fest, ob die dann anstehende Drehung nach rechts oder nach links erfolgen soll. Im Gegensatz zur letzten Version der Prozedur **`spirolaterale`** bevorzuge ich hier eine Listenform mit [1 -1], bei der dann die 1 für eine Rechtsdrehung, -1 für eine Linksdrehung steht. Dies macht die Verrechnung einfacher. Auch der Zähler wird dabei überflüssig; Abbruchkriterium ist nun, wenn die Liste **`laengen`** abgearbeitet und damit leer geworden ist. Die entsprechenden Prozeduren sehen dann so aus:

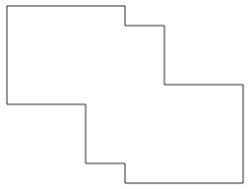
[9] Bei den Angaben von Krawczyk ist zu beachten, dass er in seinen Beschreibungen immer Innenwinkel angibt, während die Schildkrötengrafik mit Außenwinkeln arbeitet. Wenn Krawczyk also z.B. Drehungen von 60^0 angibt, sind hier (180^0 - 60^0), also 120^0 anzugeben! Außerdem wird bei Krawczyk immer die Drehung vor der Linienzeichnung ausgeführt, was bei Richtungsänderungen (Linksdrehungen) natürlich zu ganz anderen Ergebnissen führt. Wir folgen dagegen auch hier der Darstellung bei Abelson & diSessa (1981, S. 37 ff.) bzw. Fisher & Campbell (1991).

```
Prozedur spirolaterale laenge winkel laengen richtungen
spiro :laenge :winkel :laengen :richtungen
if  (and (abs xpos) < 1 (abs ypos) < 1) [stop]
[spirolaterale :laenge :winkel :laengen :richtungen]
Prozedur spiro laenge winkel laengen richtungen
if empty? :laengen [stop]
   [forward :laenge*first :laengen
    right :winkel*first :richtungen
    spiro :laenge :winkel butfirst :laengen butfirst :richtungen]
```

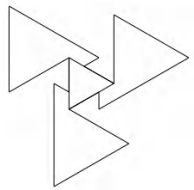
Mit `spirolaterale 20 90 [1 2 3 4 5 6] [1 1 1 1 1 1]` kann z.B. die Grundfigur mit n=6 (oben links) reproduziert werden. Mit `spirolaterale 20 90 [1 2 3 4 5 6] [-1 1 -1 1 1 1]`, also mit zwei Linksdrehungen, kommt die Figur oben Mitte heraus und bei einer Winkeländerung auf 120°, d.h. `spirolaterale 20 120 [1 2 3 4 5 6] [-1 1 -1 1 1 1]` ergibt sich die Figur oben rechts.



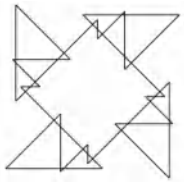
spirolaterale 20 90 [1 2 3 4 5 6] [1 1 1 1 1 1]



spirolaterale 20 90 [1 2 3 4 5 6] [-1 1 -1 1 1 1]

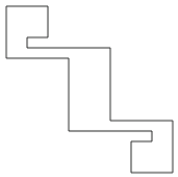


spirolaterale 20 120 [1 2 3 4 5 6] [-1 1 -1 1 1 1]



spirolaterale 20 135 [1 2 3 4 5 6] [-1 1 -1 -1 -1 1]

spirolaterale 20 90 [1 2 3 4 5 6 7] [1 1 -1 -1 -1 1 1]



spirolaterale 20 90 [1 2 3 4 5 6 7 8] [1 -1 -1 -1 -1 1 -1 1]

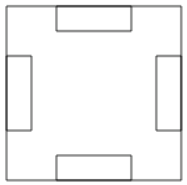
Krawczyk hat in *Spirolaterals, complexity from simplicity* (1999) sowie in *The Art of Spirolaterals* (2000) entsprechende Varianten vorgestellt, z.B. mit `spirolaterale 20 135 [1 2 3 4 5 6] [-1 1 -1 -1 -1 1]` die Figur unten links.

Auch in dem Artikel *Investigating Spirolaterals through LOGO* von Fisher & Campbell (1991) finden sich sowohl mathematische Hintergründe und viele Beispiele, die mit der hier erreichten Variabilität leicht nachvollzogen werden können (vgl. die obere Reihe der Abbildung S. 40). So führen sie zusätzlich negative Zahlen für die Längen (Rückwärtsbewegung der Turtle) ein, was erhebliche Auswirkungen auf das Erscheinungsbild hat (vgl. dazu die Variationen in der mittleren Reihe).

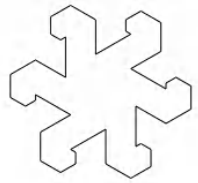
Fisher & Campbell führen eine weitere Notation ein, um spezielle Listen gewünschter Längen zu komprimieren. Diese sollen allgemein folgende Form haben: [1 1 0 1 0 0 ... 1 0 ... 0 1 0 ... 0 1 0 0 1 0 1]. Eine Null bedeutet also, dass eine Drehung am Platz ausgeführt wird, die Schildkröte also vor der nächsten Bewegung sich (je nach Anzahl der Nullen) um das Doppelte bzw. Mehrfache des Winkels gedreht hat. In der von Fisher & Campbell eingeführten Schreibweise bedeutet dann die 2 eine [1 0]-Folge, die 3 eine [1 0 0]- Folge usw. Von `spirolaterale` wird die neue Prozedur `aspiro` aufgerufen, die folgendes Aussehen hat:

```
Prozedur aspiro laenge winkel laengen
if empty? :laengen [stop]
     [forward :laenge right :winkel make "test first :laengen
if  :test > 1 [make "n first :laengen right :winkel * (:n - 1)] []
aspiro :laenge :winkel butfirst :laengen]
```

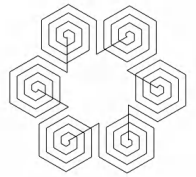
Ergebnis sind Gebilde wie in der untersten Reihe der Abbildung.



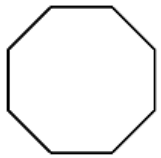
spirolaterale 20 90 [1 5 5 1 3] [1 1 1 1 1]



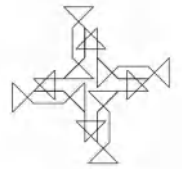
spirolaterale 20 60 [1 2 3 4 5 6 -7] [1 1 1 1 1 1 1]



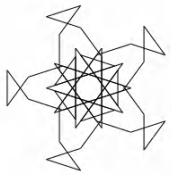
spirolaterale 2 60 [1 2 3 ... 24 -25] [1 1 1 ... 1 1 1]



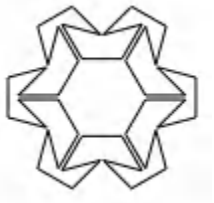
spirolaterale 40 45 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1] [1 ... 1]



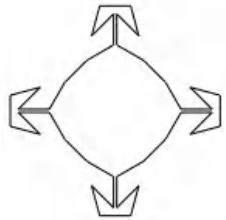
spirolaterale 40 45 [1 -1 1 1 -1 1 1 -1 1 1 1 1 -1] [1 ... 1]



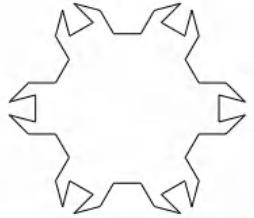
spirolaterale 40 36 [1 -1 1 1 -1 1 1 1 -1 1 1 1 1 -1] [1 ... 1]



spirolaterale 20 75 [1 2 3 4 4 3 2 1]



spirolaterale 20 75 [1 2 3 4 5 5 4 3 2 1]



spirolaterale 20 105 [1 2 3 4 4 3 2 1]

Rekursive Grafiken

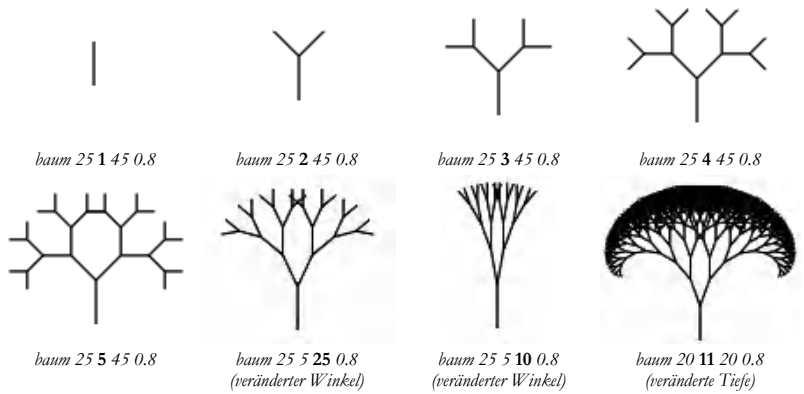
Die Mächtigkeit der Rekursion haben wir bereits mehrfach kennen gelernt. Es gibt eine ganze Klasse von Grafiken, die auf diesem Prinzip aufbauen. Das durchgängige Prinzip der rekursiven Grafiken soll an einer der bekanntesten dieser Grafiken erläutert werden, dem rekursiven Baum, der fast so etwas wie ein Symbol der Logo-Rekursion geworden ist (Abelson & DiSessa, 1981, S. 82 f.).

Rekursive Bäume

Die entsprechende Prozedur zeichnet einen Stamm, an dessen Ende zwei kleinere Äste ansetzen. Jeder Ast ist nun seinerseits Träger zweier weiterer kleiner Äste. Dies wird bis zu einer vorgegebenen Rekursionstiefe fortgesetzt. Um das Ganze gleich so variabel wie möglich zu halten, wird auch ein Parameter `winkel` eingeführt, mit dem die Äste vom Stamm abzweigen, sowie ein Parameter `verkuerzung`, der angibt, wie die Äste pro Rekursionstiefe schrumpfen.

```
Prozedur baum seite tiefe winkel verkuerzung
if :tiefe = 0 [stop]
[forward :seite
left :winkel
baum (:seite * :verkuerzung) (:tiefe - 1) :winkel :verkuerzung
right 2 * :winkel
baum (:seite * :verkuerzung) (:tiefe - 1) :winkel :verkuerzung
left :winkel back :seite]
```

Damit zum internen Ablauf der Prozedur. Sie wird abgebrochen, wenn die Tiefe Null erreicht ist (alternativ könnte auch ein Unterschreiten eines Minimums der Astlänge als Abbruchkriterium heran gezogen werden). Dann wird zunächst der Stamm gezeichnet. Es folgt der Aufruf der beiden Äste, erst der linke, dann der rechte Ast. Am Ende erfolgt eine Linksdrehung und eine



baum 25 **1** *45 0.8*

baum 25 **2** *45 0.8*

baum 25 **3** *45 0.8*

baum 25 **4** *45 0.8*

baum 25 **5** *45 0.8*

baum 25 5 **25** *0.8*
(veränderter Winkel)

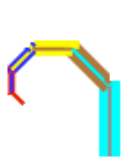
baum 25 5 **10** *0.8*
(veränderter Winkel)

baum 20 **11** *20 0.8*
(veränderte Tiefe)

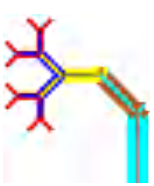
Rückwärtsbewegung, damit die Schildkröte zum Ausgangspunkt zurück kommt. Der so definierte Mechanismus lässt sich an den Bäumen unterschiedlicher Tiefe nachvollziehen.

Der Erhalt der Ausgangsposition ist eine nützliche Eigenschaft vieler rekursiver Prozeduren. Prüfen Sie einfach den Effekt, wenn diese Eigenschaft fehlt, diese Befehle also heraus genommen werden.

Erst bei großer Tiefe (etwa > 10) wird wegen der Zeichengeschwindigkeit richtig deutlich, wie der Baum gezeichnet wird. Aufgrund der rekursiven Struktur wird immer zuerst der linke Ast gezeichnet und zwar auf allen Stufen. In der folgenden Abbildung ist dies in einigen Stadien verdeutlicht (verstärkt durch farbliche Markierung der Stadien und abnehmende Dicke mit zunehmender Tiefe). Das entspricht allerdings kaum unserem Verständnis davon, wie ein realer Baum wachsen würde.

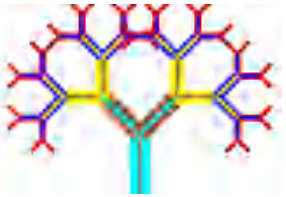


baum 40 6 45 0.75
linke Äste, Tiefe 6



baum 40 6 45 0.75
linke Äste, Tiefe 4-6

baum 40 6 45 0.75
gesamte linke Seite



baum 40 6 45 0.75
gesamter Baum

Trotzdem ergeben sich bei geeigneter Wahl der Parameter Baumstrukturen, die realen Bäumen bestimmter Arten sehr ähneln.

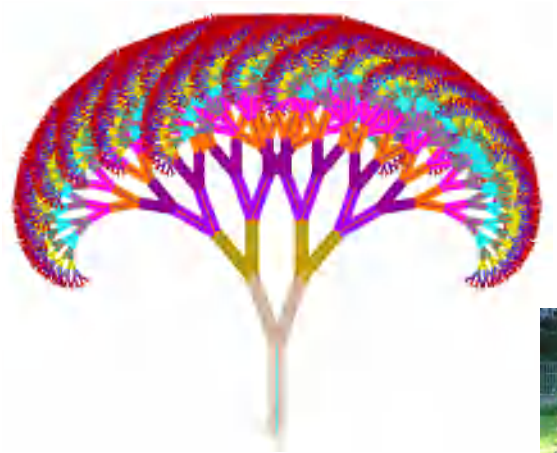
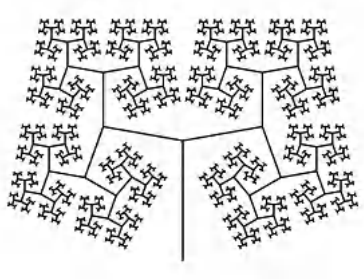


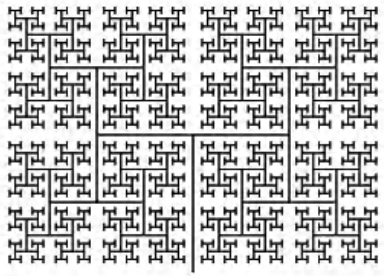
Foto: Alter Botanischer Garten Tübingen

Mit kleinen Erweiterungen können Sie weitere Varianten erstellen, z.B. Bäume, die von konstantem „Ostwind" oder „Westwind" entsprechende Schieflage bekommen haben. Auch zufällige Änderungen der Winkel und der Längen können den Realismus der Bäume erhöhen. Achten Sie dabei immer auf die korrekte Rückführung der Schildkröte auf die Ausgangspositionen.

Mit anderen Parameterwerten ergeben sich eher abstrakte Figuren, die bei entsprechender Wahl der Verkürzung fast flächenfüllende Muster darstellen.



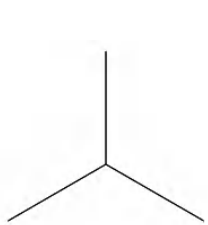
baum 100 11 **80** *0.67*

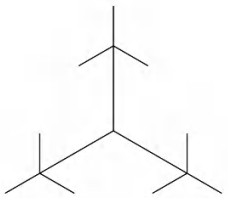


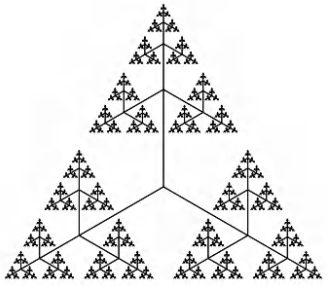
baum 100 11 **90** *0.7*

Bei Lauwerier (1989) habe ich eine Erweiterung des rekursiven Baumes gefunden, die von ihm ternärer Baum genannt wird, weil die Struktur auf dem Ternärsystem beruht. Von einem Punkt gehen drei Äste jeweils im Winkel von 120° aus. Jedes der Enden ist seinerseits Ausgangspunkt von drei kleineren Ästen in den gleichen Richtungen. Bei Lauwerier (1989, S. 23) findet sich der mathematische Bezug zu ternären Brüchen. Für uns reicht es, die resultierende Struktur zu erzeugen (wie in der folgenden Abbildung zu sehen). Die Befehlsabfolge bewirkt das Zeichnen des ersten Astes, den ersten zugehörigen rekursiven Aufruf, der gefolgt wird von der Rückführung der Schildkröte an den Ausgangspunkt, dann die Drehung um 120° und schließlich die zweimalige Wiederholung des Ganzen.


```
Prozedur t_baum seite tiefe verkuerzung
if :tiefe = 0 [stop]
[forward :seite
t_baum (:seite * :verkuerzung) :tiefe - 1 :verkuerzung
back :seite right 120 forward :seite
t_baum (:seite * :verkuerzung) :tiefe - 1 :verkuerzung
back :seite right 120 forward :seite
t_baum (:seite * :verkuerzung) :tiefe - 1 :verkuerzung
back :seite  right 120]
```





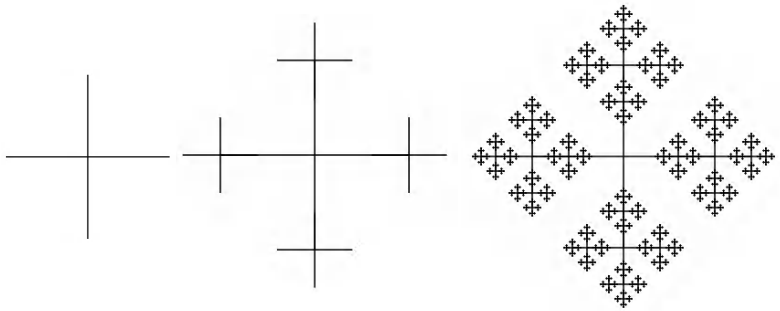


t_baum 100 **1** *0.47* *t_baum 100* **2** *0.47* *t_baum 100* **7** *0.47*

Wie vermutlich schon zu bemerken war, habe ich eine Vorliebe für Verallgemeinerungen. Warum also nicht gleich eine Prozedur, bei der beliebig viele Äste abzweigen können? Damit Sie sehen, wie einfach diese Verallgemeinerung zu erreichen ist, finden Sie unten die Prozedur für einen n-Baum, hier `bueschel` genannt (weil eine Baumstruktur eigentlich nicht mehr zu erkennen ist).

Die Veränderungen gegenüber der Prozedur `t_baum` sind fett markiert. Letztlich wird dadurch sogar eine Verkürzung der Prozedur erreicht. In der zugehörigen Abbildung sind einige Konfigurationen und zum Abschluss eine Figur mit Farben dargestellt, die den Büschel-Charakter der Grafiken noch verdeutlicht.

```
Prozedur bueschel seite tiefe verkuerzung n
if :tiefe = 0 [stop]
[forward :seite
repeat (:n - 1)
[bueschel (:seite * :verkuerzung) :tiefe - 1 :verkuerzung :n
back :seite right 360/:n forward :seite]
bueschel (:seite * :verkuerzung) :tiefe - 1 :verkuerzung :n
back :seite  right 360/:n]
```

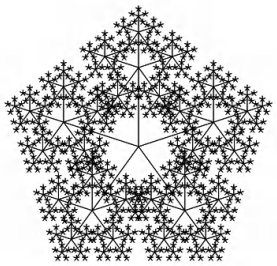


bueschel 100 **1** *0.4* **4**

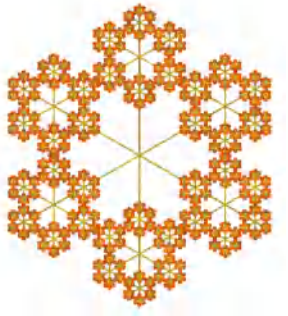
bueschel 100 **2** *0.4* **4**

bueschel 100 **6** *0.4* **4**

bueschel 100 **2** *0.45* **5**



bueschel 100 **5** *0.45* **5**



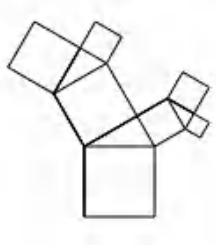
bueschel 100 **6** *0.35* **6**

Pythagoras-Baum

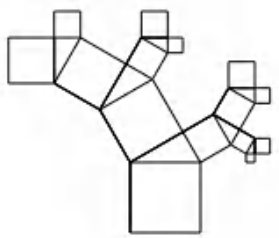
Eine weitere Baumstruktur ist der Pythagoras-Baum. Das ursprüngliche Verfahren zum Erstellen eines Pythagoras-Baums basiert auf dem Satz des Pythagoras. Gegeben ist ein Quadrat mit einer bestimmten Seitenlänge. Über der oberen Seite zeichnet man ein rechtwinkliges Dreieck (in unserem Beispiel hat dieses die Winkel 30^0 und 60^0). Über die noch freien Katheten zeichnet man die Kathetenquadrate. So entsteht die bekannte Darstellung zum Pythagoräischen Lehrsatz (vgl. die Figur mit `pythagorasbaum 40 1`). Durch rekursives Aufrufen dieser Konstruktionsvorschrift wird eine Figur erzeugt, die im Grenzfall wieder der Form eines unsymmetrischen Baumes ähnelt (vgl. unten mit Tiefe 12).

```
Prozedur pythagorasbaum basis tiefe
repeat 4[forward :basis right 90]
if :tiefe = 0 [stop]
[forward :basis left 45
pythagorasbaum (sqrt 2) * :basis/2 :tiefe - 1 right 90 forward
(sqrt 2) * :basis/2
pythagorasbaum (sqrt 2) * :basis/2 :tiefe - 1
back (sqrt 2) * :basis/2 left 45 back :basis]
```

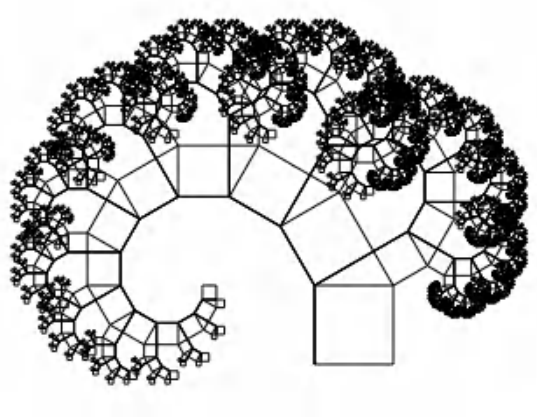

pythagorasbaum 40 **1**



pythagorasbaum 40 **2**



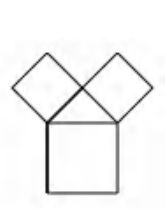
pythagorasbaum 40 **3**



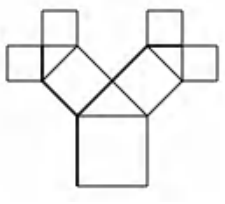
pythagorasbaum 40 **12**

Eine Variante des Pythagoras-Baums ist die mit einem gleichseitigen Dreieck, d.h. mit den beiden Winkeln von 45^0. Es entsteht dann ein symmetrischer Baum. Die zugehörige Prozedur ist nur verändert hinsichtlich der Winkelvorgaben und der Berechnung der Seitenlängen (fett markiert). Das Ergebnis bei höheren Tiefen ist wieder ein eher abstraktes Muster (vgl. unten mit Tiefe 12).

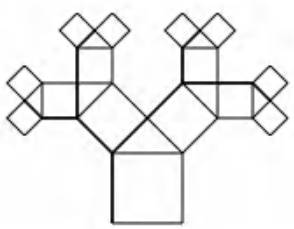
```
Prozedur pythagorasbaum_45 basis tiefe
repeat 4[forward :basis right 90]
if :tiefe = 0 [stop]
[forward :basis left 45
pythagorasbaum_45 (sqrt 2) * :basis/2 :tiefe - 1 right 90 forward
(sqrt 2) * :basis/2
pythagorasbaum_45 (sqrt 2) * :basis/2 :tiefe - 1
back (sqrt 2) * :basis/2 left 45 back :basis]
```



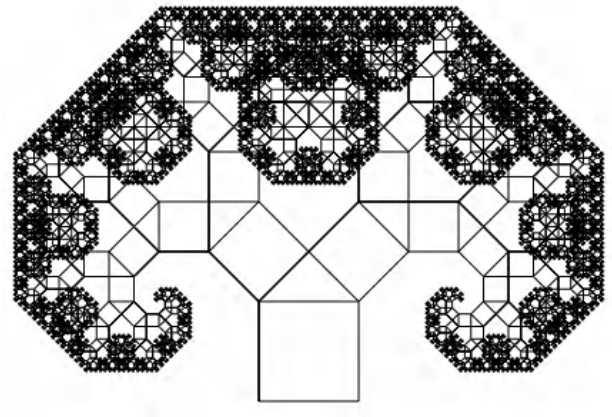
pythagorasbaum_45 40 **1**



pythagorasbaum_45 40 **2**



pythagorasbaum_45 40 **3**



pythagorasbaum_45 40 **12**

L-Systeme

In diesem Kapitel stehen spezielle rekursive Grafiken im Mittelpunkt, die sogenannten Lindenmayer-Systeme oder L-Systeme. Sie wurden 1968 von dem ungarischen Biologen Aristid Lindenmayer eingeführt. Er wollte damit eine axiomatische Theorie der biologischen Entwicklung vorlegen, Prozesse, die er bis dahin an einfachen mehrzelligen Organismen studiert hatte. Später wurde dieser Ansatz ausgeweitet auf höhere Pflanzen und komplexe Verzweigungsstrukturen. Inzwischen haben die L-Systeme in der Computergrafik Verwendung gefunden bei der realitätsnahen Modellierung von Pflanzen sowie bei der Erzeugung von Fraktalen. Eine grundlegende Einführung mit schönen Bildbeispielen findet sich in Prusinkiewicz & Lindenmayer (1990).

Es handelt sich bei den L-Systemen allgemein gesprochen um ein Ersetzungssystem, das auf Zeichenketten basiert. Solche Systeme werden auch als formale Grammatiken beschrieben und behandelt. Das System enthält eine Menge von Buchstaben, die gewissermaßen das Alphabet des Systems darstellen. Dazu gibt es einen Satz von Ersetzungsregeln (die Generatoren), die festlegen, durch welche Buchstaben bzw. Buchstabenfolgen ein Buchstabe ersetzt wird. Begonnen wird mit einer Startsequenz von Buchstaben (dem Initiator). Die Regeln werden dann rekursiv angewandt.

Die einfachsten dieser Systeme werden als D0L-Systeme bezeichnet (D steht für deterministisch, 0L für 0-Kontext, d.h. kontextfreies System). Anders als in den vorherigen Kapiteln soll hierfür zunächst ein Beispiel ohne Grafik den Ansatz verdeutlichen - eine Prozedur zu den Fibonacci-Zahlen (ein Beispiel, das sich auch bei Prusinkiewicz & Lindenmayer, 1990, S. 3 f., findet).

In diesem konkreten Beispiel benötigen wir nur die beiden Buchstaben A und B. Wir starten mit einem Initiator A und den Ersetzungsregeln, dass aus A ein B (Generator A) entsteht und aus B die Buchstabenfolge AB (Generator B). Diese Vorgaben finden sich in der Prozedur `fibonacci`. In deren Zentrum steht dann eine Wiederholungsschleife, die mit einer leeren Zeichenkette `beginnsequenz` beginnt. In der Prozedur `sequenz` wird diese Zeichenkette sukzessive von der mit den Ersetzungsregeln erzeugten Zeichenkette ersetzt und ausgedruckt. Es entsteht die in

```
Prozedur fibonacci tiefe
```

```
make "initiator "A
make "generatorA "B
make "generatorB "AB
make "fibonaccisequenz :initiator
(print count :fibonaccisequenz  :fibonaccisequenz)
repeat :tiefe [make "beginnsequenz " sequenz :fibonaccisequenz
(print count :fibonaccisequenz  :fibonaccisequenz)]
```

```
Prozedur sequenz zeichenkette
```

```
if :zeichenkette = " [make "fibonaccisequenz :beginnsequenz stop]
[make "befehl (first :zeichenkette)
if (:befehl = "A) [make "beginnsequenz word :beginnsequenz :genera-
torA] []
if (:befehl = "B) [make "beginnsequenz word :beginnsequenz :genera-
torB] []
sequenz butfirst :zeichenkette]
```

der folgenden Tabelle wiedergegebene Abfolge, wobei die Länge der entstehenden Zeichenketten der berühmten Fibonacci-Folge entspricht.

Tiefe	Länge	Sequenz
0	1	A
1	1	B
2	2	AB
3	3	BAB
4	5	ABBAB
5	8	BABABBAB
6	13	ABBABBABABBAB
7	21	BABABBABABBABBABABBAB
8	34	ABBABBABABBABBABABBABABBABABBABABBAB
9	55	BABABBABABBABBABABBABABBABABBABABBABABBABABBABABBABABBABABBAB

Koch-Kurven

Die grafischen L-Systeme werden dadurch gewonnen, dass den Buchstaben Schildkröten-Interpretationen gegeben werden. Wir beginnen mit den folgenden Regeln:

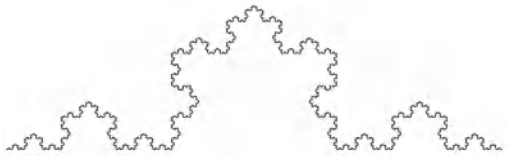
V VOR: Gehe einen Schritt nach vorn und zeichne dabei eine Linie (gegebener Länge)
P PLUS: Drehe dich (um gegebenen Winkel) nach links
M MINUS: Drehe dich (um gegebenen Winkel) nach rechts

Wir verwenden diese Regeln zur Erzeugung der Koch-Kurve. Es handelt sich bei ihr um eines der ersten formal beschriebenen fraktalen Objekte. Die von dem Mathematiker Helge von Koch 1904 vorgestellte Kurve wurde bei ihrer Entdeckung auch als *Monsterkurve* bezeichnet. Die Konstruktion erfolgt in folgendem iterativen Prozess:

- Zeichne eine Strecke gegebener Länge
- Teile die Strecke in drei gleiche Teile
- Zeichne über der mittleren Strecke ein gleichseitiges Dreieck
- Wiederhole dieses Vorgehen mit jeder der nun vier Strecken

Die Konstruktion führt bei bis zu drei Iterationen zu folgenden Kurven:

tiefe **0** *tiefe* **1** *tiefe* **2** *tiefe* **3**



tiefe **6**

Um diese Konstruktion als L-System durchzuführen ist nun der Initiator zu definieren, hier also V (die gerade Ausgangsstrecke) und der Generator, hier die Abfolge VPVMMVPV (das gleichseitige Dreieck). Der Initiator wird in der Prozedur `polygon` abgearbeitet, der Generator in der

```
Prozedur kochkurve
make "initiator "V
make "generator "VPVMMVPV
make "winkel 60
make "seite 4
make "tiefe 4
make "restinitiator :initiator
polygon
```

```
Prozedur polygon
if (:restinitiator = '') [stop]
[make "befehl (first :restinitiator)
make "restinitiator butfirst :restinitiator
if (:befehl = "M) [M] []
if (:befehl = "P) [P] []
if (:befehl = "V) [umweg :generator :tiefe] []
polygon]
```

```
Prozedur umweg nochgenerator tiefe
if (:nochgenerator = '') [stop]
[make "befehl first :nochgenerator
make "nochgenerator butfirst :nochgenerator
if (:befehl = "M) [M] []
if (:befehl = "P) [P] []
if (and :befehl = "V :resttiefe = 0) [V stop] []
if (and :befehl = "V :resttiefe > 0) [umweg :generator (:resttiefe
- 1)] []
umweg :nochgenerator :resttiefe]
```

```
Prozedur V
forward :seite
```

```
Prozedur P
left :winkel
```

```
Prozedur M
right :winkel
```

Prozedur `umweg`. Diese beiden Prozeduren rufen jeweils die Befehle `M`, `P` und `V` auf. Zu beachten ist noch, dass jeweils die nach jeder Iteration verbleibenden Buchstabenfolgen zwischengespeichert werden müssen.

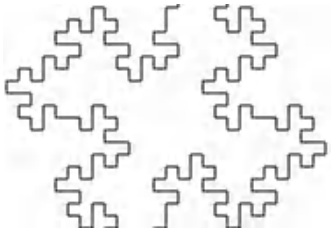
Die Prozeduren `M`, `P` und `V` wären eigentlich überflüssig, wenn deren Befehle bei den jeweiligen Abfragen direkt ausgeführt würden. Sie sind aber wegen der Anlehnung an das Regelsystem in eigene Prozeduren verlagert.

Es ist nun sehr einfach die unterschiedlichsten Koch-Kurven zu erzeugen. Es reicht, den Initiator, den Generator, den Winkel und die gewünschte Tiefe festzulegen. Die folgenden Beispiele haben alle einen konstanten Winkel von 90^0. Sie reproduzieren die bei Prusinkiewicz & Lindenmayer (1990, S. 8 ff.) gezeigten Kurven.

In der ersten Zeile wird eine quadratische Koch-Insel (laut Mandelbrot auch Minkowski-Kurve genannt) mit der Tiefe 0, 1 und 2 gezeigt, die noch einmal die Entstehung verdeutlicht. Sie verwendet den Initiator `VMVMVMV` (wie übrigens alle folgenden Koch-Kurven auch) und den Generator `VMVPVPVVMVMVPV`.

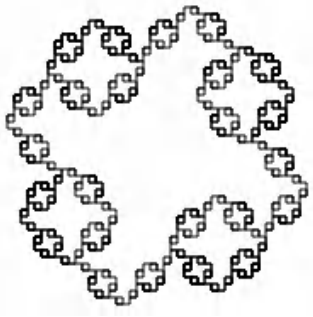
seite 60
tiefe 0

seite 20
tiefe 1

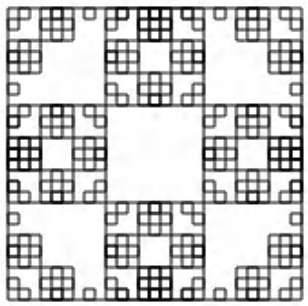


seite 7
tiefe 2

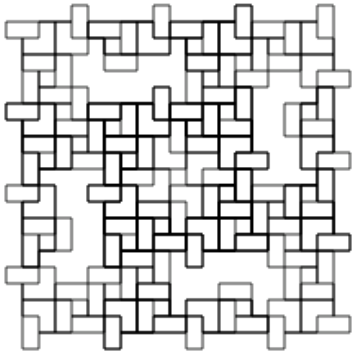
Im Folgenden wird aus Platzgründen nur noch der Generator angegeben. Das verdeutlicht gleichzeitig, wie einfach mit den allgemein gehaltenen Prozeduren die L-Systeme erzeugt werden können.



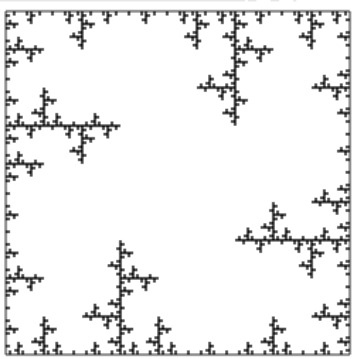
VVMVMVMVMVMVPV



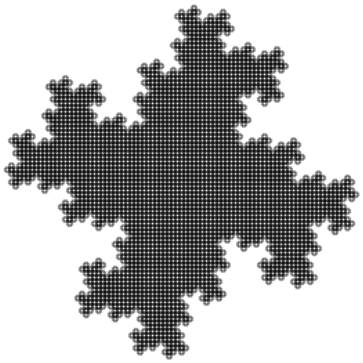
VVMVMVMVMVV



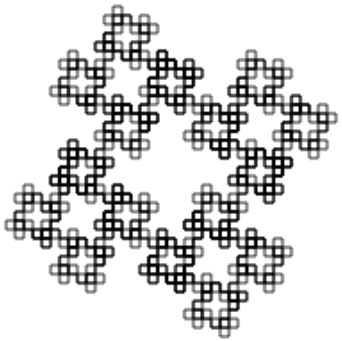
VVMVPVMVMVV



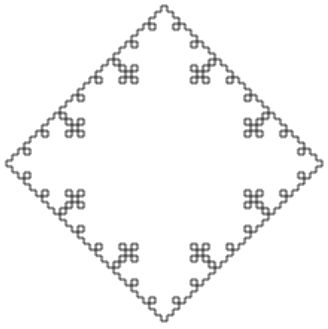
VVMVMMVMV



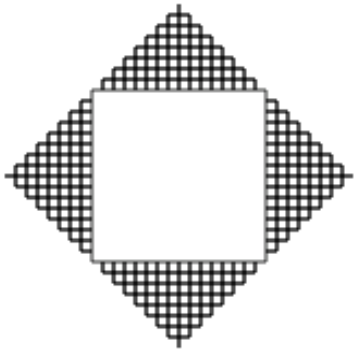
VMVVMMVMV



VMVPVMVMV



VPVMVMVPV

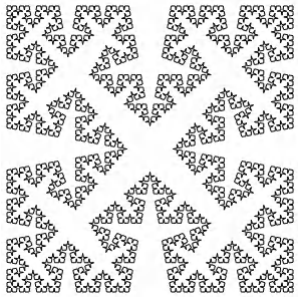


VPVMMVPV

Eine Vielzahl weiterer Varianten der Koch-Kurve sind möglich und führen zu interessanten Formen, die hier aus Platzgründen nicht selbst dargestellt werden sollen (nähere Informationen dazu in Lauwerier, 1989, S. 47 ff.). Sie können mit folgenden Vorschlägen weiter experimentieren (die Angaben beschränken sich jeweils auf Initiator, Generator, Winkel und Tiefe):

Kochinsel Dreieck (nach innen): Initiator VMMVMMV, Generator VMVPPVMV, Winkel 60, Tiefe 3
Kochinsel Quadrat (nach innen): Initiator VMMMVMMMVMMMV, Generator VMMVPPPPVMMV, Winkel 30, Tiefe 4
Kochinsel Fjord: Initiator VMMVMMV, Generator VMVVPPVVVVMMVVPV, Winkel 60, Tiefe 3
Muschelkurve (auch Lévy-Kurve genannt): Initiator PVPV, Generator VPVM, Winkel 90, Tiefe 10
Lévy-Teppich: Initiator VMMVMMVMMV, Generator PVMMVP, Winkel 45, Tiefe 10

Koch-Kurve und Koch-Insel unterschieden sich lediglich im Basiswinkel (60^0 bzw. 45^0). Bei schließlich beliebigen Basiswinkeln sprechen wir von der Cesàro-Kurve. Bei der Wahl des Basiswinkels ist allerdings immer darauf zu achten, dass eine Winkelsumme von 180^0 erreicht wird. Im hier gezeigten Beispiel handelt es sich um ein Cesàro-Quadrat (bei dem die Form nach innen zeigt). Gewählt wurde ein Basiswinkel von 10^0 und eine Tiefe von 5. Der Initiator für das Quadrat wird dadurch etwas länglich (ebenso wie der Generator) *VMMMMMMMMMV-MMMMMMMMMVMMMMMMMMV*.



VMMMMMMMMMVPPPPPPPPPPPPPPPVMMMMMMMMV

Als nächstes erweitern wir den bisherigen Regelsatz um folgende Regel:

S SPRUNG: Gehe einen Schritt nach vorn ohne dabei eine Linie zu zeichnen

Dafür ist ein neuer Generator S zu definieren, der in einer neuen Prozedur `sprung` abgearbeitet wird, die dann ihrerseits den Befehl S aufruft.

```
Prozedur sprung restgenerator resttiefe
```

```
if (:restgenerator = '') [stop]
[make "befehl first :restgenerator
make "restgenerator butfirst :restgenerator
if (and :befehl = "S :resttiefe = 0) [S stop] []
if (and :befehl = "S :resttiefe > 0) [sprung :generatorS (:resttie-
fe - 1)] []
sprung :restgenerator :resttiefe]
```

```
Prozedur S
```

```
penup forward :seite pendown
```


Zur Verdeutlichung der damit gegebenen Möglichkeiten betrachten wir die Cantor-Menge. Sie entsteht, indem aus einer Einheitsstrecke das mittlere Drittel entfernt wird, dann vom ersten und dritten Drittel wieder das mittlere Drittel und so weiter. Die grafische Darstellung ergibt als einfachstes Beispiel einer fraktalen Kurve den Cantor-Staub (zur Normierung der Darstellung ist dabei die Strecke auf jeder Rekursionsstufe zu dritteln). Die zugehörige Prozedur `cantor` entspricht ansonsten der bereits bekannten Prozedur für die Koch-Kurve.

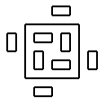
```
Prozedur cantor
```

```
make "initiator "V
make "generatorV "VSV
make "generatorS "SSS
make "seite 120 make "tiefe 0
make "restinitiator :initiator
polygon
```

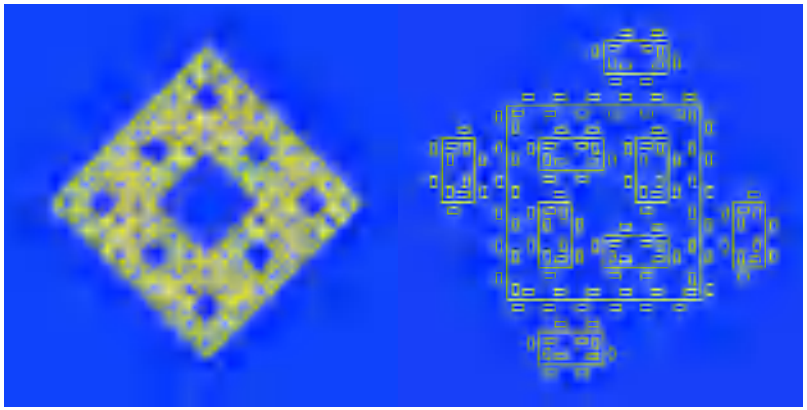
cantor mit tiefe 5

Ebenfalls durch „Springen“ entsteht der Sierpinski-Teppich. Die Grundform bildet aus der Einheitsstrecke ein Rechteck mit angehängten Strecken (unten links). Bei entsprechender Tiefe der Rekursion entsteht dann der Teppich. Das „Springen“ kennzeichnet auch eine Variante der Koch-Kurve, die zu kombinierten Koch-Inseln und Koch-Seen führt (unten rechts).

Grundform Sierpinski-Teppich:
initiator V
generatorV VPVMVMVVMVMVMSV
generatorS SSS
tiefe 1



Grundform Koch-Inseln:
initiator VPVPVPV
generatorV VPSMVVPVPVVPVSPVVMSPVVMVMVVMVSMVVV
generatorS SSSSSS
tiefe 1



Sierpinski-Teppich *Koch-Inseln mit Koch-Seen*

Allein mit den vier Regeln V, P, M und S lassen sich also eine Vielzahl von L-Systemen darstellen. Sie können damit entsprechend freizügig experimentieren. Natürlich können Sie selber weitere Regeln definieren; Anregungen dazu liefert z.B. Prusinkiewicz & Lindenmayer (1990). Einfach umzusetzen ist etwa, die Farbe zu ändern.

Zitierte Literatur

Abelson, H. (1983). Einführung in Logo. München: IWT-Verlag.

Abelson, H. & diSessa, A.A. (1981) Turtle Geometry: The Computer as a Medium for Exploring Mathematics. Cambridge, Mass.: The MIT Press

Böcker, H.-D., Fischer, G. & Plehnert, M. (1986). Interaktives Probemlösen mit LOGO Band 1: Einführung in das interaktive Programmieren. München: IWT-Verlag.

Böcker, H.-D., Fischer, G. & Schollwöck, U. (1987). Interaktives Probemlösen mit LOGO Band 2: Praktische Projekte (Mathematik, Informatik, Künstliche Intelligenz und Sprache, Spiele). München: IWT-Verlag.

Boytchev, P. (2014). Logo Tree Project. Rev. 2.09. Download unter http://www.elica.net/download/papers/logotreeproject.pdf

Briggs, J. (2013). Python kinderleicht! Heidelberg: dpunkt Verlag.

Bundy, A. (1986). Praktische Einführung in die Künstliche Intelligenz – Mit Programmbeispielen in LOGO und LISP. München: IWT.

Chakraborty, A., Graebner, R. & Stocky, T. (1999). Logo - a Project History. Download unter http://web.mit.edu/6.933/www/LogoFinalPaper.pdf

Ducasse, S. (2005). Squeak: Learn programming with Robots. Berkeley: Apress Publishers. Download unter https://gforge.inria.fr/frs/download.php/10764/BotsInc-OriginalEnglish.pdf

Eshel, E. (2011). Kids Can Program Too!: Java Edition. CreateSpace Independent Publishing.

Fisher , W. & Campbell, R. (1991). Investigating Spirolaterals through LOGO. The College Mathematics Journal, Vol. 22, No. 2. (Mar., 1991), pp. 148-159. Download unter http://links.jstor.org/sici?sici=0746-8342%28199103%2922%3A2%3C148%3AISTL%3E2.0.CO%3B2-4

Dörner, D. (1976, 1979). Problemlösen als Informationsverarbeitung. Stuttgart: Kohlhammer.

Harvey, B. (1997). Computer Science Logo Style Vol. 1: Symbolic Computing. Cambridge, Mass.: The MIT Press. Online-Version unter http://www.cs.berkeley.edu/~bh/v1-toc2.html

Harvey, B. (1997). Computer Science Logo Style Vol. 2: Advanced Techniques. Cambridge, Mass.: The MIT Press. Online-Version unter http://www.cs.berkeley.edu/~bh/v2-toc2.html

Harvey, B. (1997). Computer Science Logo Style Vol. 3: Beyond Programming. Cambridge, Mass.: The MIT Press. Online-Version unter http://www.cs.berkeley.edu/~bh/v3-toc2.html

Hoppe, H.-U. (1984). LOGO im Mathematikunterricht. München: IWT-Verlag.

Hoppe H.-U. & Löthe, H. (1984). Problemlösen und Programmieren mit Logo. Ausgewählte Beispiele aus Mathematik und Informatik. Stuttgart: Teubner.

Hromkovič, J. (2010). Einführung in die Programmierung mit Logo. Wiesbaden: Vieweg + Teubner. Gekürzt als Skript beim ABZ für Informatikunterricht der ETH Zürich erhältlich. Download unter http://www.abz.inf.ethz.ch/media/archive1/unterrichtsmaterialien/primarschulen_stufe_sek_1/logo_heft_de.pdf

Kohler, T., Spannagel, C. & Klaudt, D. (2008). Räumliches Denken mit Logo: Eine Einführung in der Grundschule. Notes on Educational Informatics - Section B: Classroom Experiences 3 (1), pp. 29-43. Ludwigsburg: Institut für Mathematik und Informatik. Download unter https://www.ph-ludwigsburg.de/8827.html

Krawczyk, R. (2000). The Art of Spirolateral Reversals, N. Friedman (Ed.): ISAMA 2000, Conference of The International Society of the Arts, Mathematics and Architecture. Albany, New York. Download unter http://www.iit.edu/~krawczyk/isama00.pdf/

Krawczyk, R. (1999). Spirolaterals, Complexity from Simplicity. N. Friedman & J. Barrallo (Eds.). ISAMA 1999, Conference of The International Society of Arts, Mathematics and Architecture. San Sebastian, Spain. Download unter http://www.iit.edu/~krawczyk/isama99.pdf

Lauwerier, H. (1989). Fraktale verstehen und selbst programmieren. Hückelhoven: Wittig Fachbuchverlag.

Odds, F. (1973). Spirolaterals. The Mathematics Teacher 66, February 1973, pp. 121-124.

Odds, F. (2003). Brief an R. Krawczyk: Spirolaterale. Download unter http://home.netcom.com/~bitart/spirols/spdesc11.htm

Papert, S. (1982). Mindstorms: Kinder, Computer und Neues Lernen. Stuttgart: Birkhäuser.

Papert, S. & Minsky, M. (1969, 1988[2]). Perceptrons. Cambridge, Mass.: The MIT Press

Prusinkiewicz, P. & Lindenmayer, A. (1990). The algorithmic beauty of plants. New York: Springer.

Rauch, H. & Wedekind, J. (1989). Schildkrötengrafik: Einfaches Programmieren von Grafiken. Tübingen: DIFF.

Resnick , M. (1994). Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds. Cambridge, Mass.: The MIT Press.

Rushkoff, D. (2011). Program or be Programmed. OR Books.

Stieff, M.; Wilensky, U. (2002): ChemLogo – An emergent modeling environment for teaching and learning chemistry. Proceedings of the 5th biannual International Conference of the Learning Sciences. Download unter http://ccl.northwestern.edu/uri/public_html/papers/chemlogo/

Strom, C. (2014). Kids programmieren 3D-Spiele mit JavaScript. Köln: O'Reilly.

Tauber, S. (1883). Logo. In: LOGIN 3/1983, S. 37 ff. Download unter http://joachim-wedekind.de/Downloads/Logo_LOGIN3-83.pdf

Ziegenbalg, J. (1984). Programmiersprachen als Träger informatischer Grundideen. MNU 37, 7/1984, S. 404-415. Download unter http://www.ziegenbalg.ph-karlsruhe.de/materialien-homepage-jzbg/Manuskripte/Programmiersprachen-als-Traeger-von-Grundideen-der-Informatik.pdf

Ziegenbalg, J. (1985). Programmieren lernen mit Logo. München: Hanser. Download unter http://www.ziegenbalg.ph-karlsruhe.de/materialien-homepage-jzbg/mybooks-scans/Programmieren-lernen-mit-Logo.pdf

Weiterführende Literatur

Hinweis: Neben der zitierten Literatur möchte ich auf einige weiterführende Bücher hinweisen, die für Logo-Anwender viele interessante Aspekte bieten, die über die Programmierung interaktiver Grafiken hinausgehen. Leider sind einige davon schwer zu erhalten, da inzwischen vergriffen, aber Vieles ist erfreulicherweise kostenlos und online verfügbar.

Abelson, H. & Abelson, A. (1992). Logo for the Macintosh. Cambridge, Mass.: Paradigm Software.

Clayson, J. (1988). Visual Modeling with Logo. Cambridge, Mass.: The MIT Press.

Downey, A.B. & Gay, G. (2003). How to Think Like a Computer Scientist - Logo Version. Download unter http://openbookproject.net/thinkcs/archive/logo/english/thinklgo.pdf

DuCharme, B. (2002). Logo for Kids: An Introduction. Download unter http://www.snee.com/logo/logo4kids.pdf

Glaeser, G. & Polthier, K. (2009). Bilder der Mathematik. Heidelberg: Springer.

Haftendorn, D. (2010). Mathematik sehen und verstehen. Schlüssel zur Welt. Heidelberg: Spektrum Akademischer Verlag.

Joshi, A. & Gaikwad, S. (2012). Logo Programming (Part 1) [Kindle Edition]. SPARK Institute. Als E-Book erhältlich bei Amazon.

Joshi, A. & Gaikwad, S. (2012). Logo Programming (Part 2) [Kindle Edition]. SPARK Institute. Als E-Book erhältlich bei Amazon.

Lawler, R.W. (1985). Computer Experience and Cognitive Development: A Child's Learning in a Computer Culture. Chichester: Ellis HorwoodLimited.

Markus, M. (2009). Die Kunst der Mathematik. Wie aus Formeln Bilder werden. Frankfurt: Zweitausendeins.

Muller, J. (1997). The Great Logo Adventure. Doone Publications. Download unter http://www.softronix.com/download/tgla.zip

Newell, B. (1988). Turtle confusion: Logo puzzles and riddles. Canberra: The Curriculum Development Centre. Download unter http://constructingmodernknowledge.com/tcbook.pdf

Newell, B. (1988). Turtles speak mathematics. Canberra: The Curriculum Development Centre. Download unter http://constructingmodernknowledge.com/tsmbook.pdf

Papert, S. (1998). Die vernetzte Familie: Kinder und Computer. Stuttgart Kreuz.

Papert, S. (1994). Revolution des Lernens: Kinder, Computer, Schule in einer digitalen Welt. Hannover: Heise.

Peitgen, H.-O., Hürgens, H. & Saupe, D. (1992). Bausteine des Chaos. Fraktale. Stuttgart: Klett-Cotta.

Peitgen, H.-O., Hürgens, H. & Saupe, D. (1994). Chaos. Bausteine der Ordnung. Stuttgart: Klett-Cotta.

Pickover, C.A. (1991). Computers and the Imagination. Visual Adventures beyond the edge. New York: St. Martin's Press.

Stein, H. (1984). Logo. Grafik, Sprache, Mathematik. Eine Einführung in die Programmiersprache Logo unter besonderer Berücksichtigung des Apple-Logo. Haar b. München: Markt & Technik Verlag.

Watt, D. (1984). Logo. Computersprache für Eltern und Kinder. Jeder kann Programmieren! München: te-wi-Verlag.

Widmer. L. (2007). Mathematik und Geometrie mit Logo. Download unter http://www.abz.inf.ethz.ch/media/archive1/unterrichtsmaterialien/maturitaetsschulen/Mathematik_und_Geometrie_mit_Logo.pdf

Ziegenbalg, J. (1986). Informatik. Logo Lern- und Arbeitsbuch. Braunschweig: Westermann. Download unter http://www.ziegenbalg.ph-karlsruhe.de/materialien-homepage-jzbg/mybooks-scans/Logo-Lern-und-Arbeitsbuch.pdf

Links und Downloads

Erfreulicherweise gibt es neben dem hier verwendeten ACSLogo eine ganze Reihe aktueller Logo-Versionen, die gepflegt und weiter entwickelt werden. Sie sind alle geeignet für die Arbeit im schulischen Kontext. Etliche davon sind Freeware, einige dagegen nur kommerziell erhältlich. Sie werden hier vorgestellt und auf die Quellen zum Herunterladen verwiesen.

Aktuelle Logo-Versionen

Für den Apple Macintosh gibt es das leistungsfähige ACSLogo hier:
http://www.alancsmith.co.uk/logo/

Als Quasi-Standard für Logo hat sich mittlerweile das Berkeley-Logo!herauskristallisiert. Verantwortlich für desssen Entwicklung ist Brian Harvey, University of California at Berkeley. Es ist in mehreren Varianten erhältlich, nämlich für

- Windows: ftp://ftp.cs.berkeley.edu/pub/ucblogo/ucbwlogosetup.exe
- Mac OS X: http://www.cs.berkeley.edu/~bh/downloads/UCBLogo-6.0.dmg.gz
- Linux:!ftp://ftp.cs.berkeley.edu/pub/ucblogo/ucblogo.tar.gz !

Eine darauf aufbauende Version mit multimedialen Erweiterungselementen und 3D-Grafik ist MSWLogo (nur für Windows) von George Mills: http://www.softronix.com/logo.html

Davon wiederum eine Weiterentwicklung mit Elementen zur Ansteuerung externer Geräte (Messen-Steuern-Regeln) ist FMSLogo: http://fmslogo.sourceforge.net

Für die Schule ist häufig hilfreich, Sprachversionen mit deutschen Grundbefehlen verwenden zu können. Die gibt es beim MSWLogo! (http://www.ph-ludwigsburg.de/logo.html) und beim FMSLogo (http://www.lehrer.uni-karlsruhe.de/~za1880/itg/FMSLogo/).

Agentenbasierte Logo-Versionen

Mit StarLogo wurde 1991 die erste agentenbasierte Logo-Version von Mitchel Resnick eingeführt. Sie erlaubt viele Turtles - bei Bedarf mehrere Tausend - gleichzeitig per Befehl anzusprechen. Diese Turtles können untereinander und mit ihrer Umwelt interagieren. StarLogo fand einige Nachfolger; der wichtigste ist aus meiner Sicht NetLogo (2002 von Uri Wilensky vorgestellt), denn es hat einige sehr interessante Erweiterungen, wie HubNet (mit dem verteilte, interaktive Simulationen möglich werden) und eine System Dynamics Modellierkomponente, die es erlaubt, Mikro- und Makrosimulationen parallel durchzuführen. StarLogo selbst hat eine Neufassung als StarLogoTNG (The Next Generation) erlebt, mit dem 3D-Simulationswelten gebaut werden können.

- NetLogo:!http://ccl.northwestern.edu/netlogo/
- StarLogo:!http://www.media.mit.edu/starlogo/
- StarLogoTNG:!http://education.mit.edu/starlogo-tng/

Kommerzielle Logo-Versionen

Es gibt nach wie vor einige klassische Logo-Versionen, die kommerziell vertrieben werden. Sie beinhalten in unterschiedlichem Maße multimediale Erweiterungen und es gibt jeweils damit erprobte Unterrichtsmaterialien. Da die Funktionalitäten bei den Verlagen ausführlich beschrieben werden, soll deshalb jeweils ein Link dorthin ausreichen. Zu erwähnen ist, dass LCSI, von Papert 1981 mitbegründet und sich selbst *leading publisher of constructivist educational software* nennend, nach wie vor mit der Version MicroWorlds dabei vertreten ist. Auch eine Version mit deutschen Grundbefehlen gibt es von Gerhard Otte; besonders interessant u.a. deshalb, weil sie Treibersoftware für technische Interfaces enthält (z.B. Fischer-Technik).

- MicroWorlds von LCSI (Windows und Mac): http://www.microworlds.com
- ImagineLogo von Logotron (Windows): http://www.logotron.co.uk/imagine/
- TerrapinLogo (Windows und Mac): http://www.terrapinlogo.com
- Gerhard Ottes WIN-LOGO 2.0 (Windows): http://www.win-logo.de

Browserbasierte Logo-Versionen

Der Zugang zur Programmierung mit Logo wird Interessenten sehr erleichtert durch browserbasierte Versionen, die keine Installation eines Programms auf dem eigenen Rechner mehr erfordern. Sie sind meist in Lernumgebungen eingebettet, die mit Beispielen und Hilfen den direkten Einstieg erlauben (die Syntax orientiert sich wieder am Berkeley Logo):

- jslogo: http://www.calormen.com/jslogo/
- Turtle Academy: http://turtleacademy.com
- papert-logo in your browser: http://logo.twentygototen.org

Visuelle Programmierung

Mittlerweile gibt es mehrere an Logo orientierte visuelle Programmierumgebungen. Begonnen hat dies 2007 mit Scratch (entwickelt am MIT Media Lab unter Leitung von Mitchel Resnick). Programme werden dabei aus Bausteinen zusammengesetzt, die zumeist allein durch ihre Form nur das Einklinken passender Elemente erlauben und dadurch Syntax-Fehler vermeiden helfen. Daran knüpfen z.B. TurtleArt, blockly oder MIT App Inventor an. Eine interessante Weiterentwicklung ist Snap! mit mächtigen Optionen, u.a. um eigene Kontrollstrukturen zu definieren. Da es HTML5 und Javascript verwendet, läuft es vollständig im Webbrowser - auch auf mobilen Geräten.

- Scratch: http://scratch.mit.edu
- TurtleArt: http://turtleart.org
- blockly: https://code.google.com/p/blockly/
- MIT App Inventor: http://appinventor.mit.edu/explore/
- Snap!: http://snap.berkeley.edu/snapsource/snap.html

Texte und Materialien

Die Entwicklung von Logo ist an Personen gebunden, vor allem Seymour Papert und Wallace Feuerzeig sowie Harold Abelson mit ihren grundlegenden Ideen, die zur Entwicklung von Logo und dessen Funktionalitäten führten. Abelson drückte es so aus: *Logo is the name for a philosophy of education and a continually evolving family of programming languages that aid in its relization*. Das alles nachgezeichnet haben Chakraborty, Graebner & Stocky (1999) in der Studie *Logo - a Project History*. Diese Arbeit, die als schöner Einstieg in die Logo-Philosophie gelesen werden kann, gibt es als Download: http://web.mit.edu/6.933/www/LogoFinalPaper.pdf

Neben dieser Sicht von außen sollten natürlich auch die Urheber selbst zu Worte kommen. Erfreulicherweise gibt es dazu Originalquellen, die Logo Memos, die ab 1971 am MIT Artificial Intelligence Laboratory publiziert und von Andru Luvisi gesammelt und online gestellt wurden: http://www.sonoma.edu/users/l/luvisi/logo/logo.memos.html

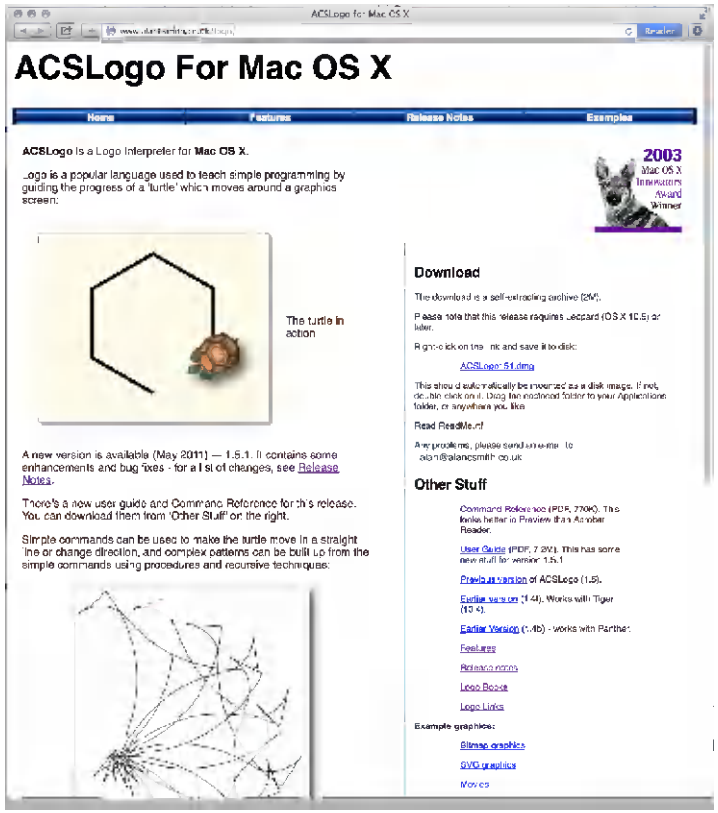
Paperts Grundideen finden sich m.E. besonders deutlich in Memo No. 2 (Teaching Chidren Thinking) und Memo No. 4 (Teaching Children to be Mathematicians vs. Teaching about Mathematics).

Mit der Logo Foundation gibt es eine Stiftung, die sich als gemeinnützige Bildungsorganisation zur Aufgabe gemacht hat, Interessenten über Logo zu informieren und sie bei der Nutzung von Logo und Logo-basierter Software und Lernumgebungen zu unterstützen. Auf ihrer Website finden sich Hinweise auf Veranstaltungen, Publikationen und Logo-Produkte: http://el.media.mit.edu/logo-foundation/

Vielleicht das Zeichen für ein kleines Logo-Revival ist die neue Webseite von Christian Spannagel zu Logo beim ZUM-Wiki. Dort finden Sie u.a. eine laufend aktualisierte Linkliste zu Logo und Literatur.

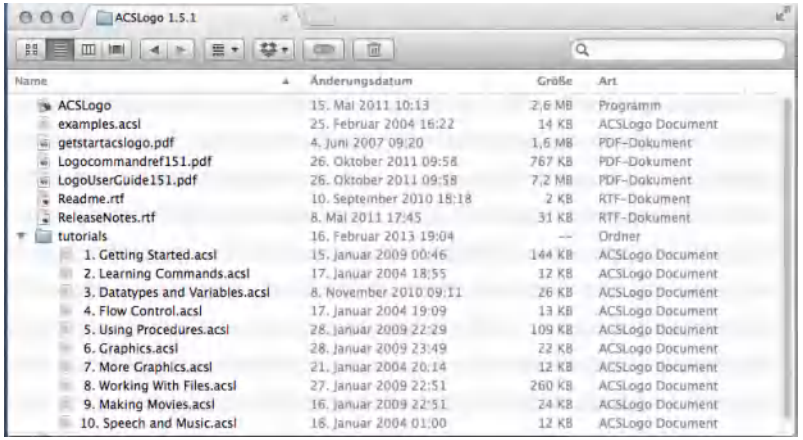
Anhang A: Installation ACSLogo

Für die Bearbeitung der Beispiele und die eigene Progarmmierung wird zuallererst natürlich das Werkzeug benötigt, also die Programmierumgebung zu ACSLogo. Diese gibt es kostenlos auf der Website des Autors Alan C. Smith: http://www.alancsmith.co.uk/logo/



Klicken Sie dort auf den Link der Image-Datei `ACSLogo151.dmg`, die Sie dann im `Download`-Ordner Ihres Rechners finden und mit Doppelklick entpacken können. Den dann vorhandenen Ordner `ACSLogo` können Sie in Gänze an einen gewünschten Ort, z.B. nach `Programme` verschieben. In dem Ordner finden Sie neben dem Programm selbst (derzeit in der Version 1.5.1) eine Logo-Datei `examples.acsl` (alle Programme werden von ACSLogo in Dateien mit der Endung .acsl abgespeichert) und einen Ordner `tutorials` mit Einführungen in die Funktionalität von ACSLogo. Diese Einführungen sind übrigens selbst in ACSLogo umgesetzt.

Sie finden auf der Website zusätzlich noch als PDF-Dateien eine Command Reference (`Logocommandref151.pdf`), mit der Beschreibung des vollständigen Befehlssatzes von ACSLogo, sowie einen User Guide (`LogoUserGuide151.pdf`), mit einer (englischen) Einführung in die Funktionalität von ACSLogo.



Wenn Sie das Programm ACSLogo gestartet haben, erscheinen zunächst zwei Fenster, ein Fenster zur Eingabe des Programmcodes (Abb. unten links), ein zweites Fenster mit der Schildkrötengrafik (Abb. unten rechts).

Im Fenster `Ohne Titel` können Sie nun direkt Logo-Befehle eingeben. Folgen Sie ab hier der weiteren Einführung im Kapitel „Die ersten Schritte“.

Anhang B: Befehlsübersicht

In der folgenden Tabelle finden Sie alle Befehle, die in ACSLogo zur Verfügung stehen, mitsamt der englischen Kurzbeschreibung. Sie ist der `Logocommandref151` entnommen, in der Sie bei Bedarf die ausführliche Beschreibung - z.T. mit Beispielen - finden.

Befehl	Funktion
+	Add
-	Subtract
*	Multiply
/	Divide
<	Less Than
>	Greater Than
ABS	Output the absolute value of a number
AND	Logical AND
Arc	Draw an Arc
ArcCosine, ArcCos	Output the angle for a Cosine
ArCosh	Inverse of Cosh
ArcSine, ArcSin	Output the angle for a Sine
ArcTangent, ArcTan	Output the angle for a Tangent
ArSinh	Inverse of Sinh
ArTanh	Inverse of Tanh
ATan2	Inverse of Tan taking two arguments
ASCII	Output a character's ASCII code
Back	Move the turtle backwards
Background, Bg	Output the background pen colour
ButFirst	Output all but the first element
ButLast	Output all but the last element
Button?, ButtonP	Output whether the left mousebutton is pressed
CanvasSize	Output the size of the canvas
Catch	Catch a Throw statement
CD	Change the Current Directory
Char	Output the character for an ASCII code
Clean	Clear the graphics window
ClearScreen, CS	Clear graphics and home the turtle
CloseReadFile	Close the file open for reading
CloseWriteFile	Close the file open for writing
ColourAtPoint, ColorAtPoint	Output the RGB values of a pixel
Cosine, Cos	Output the cosine of an angle
Cosh	Hyperbolic cosine
Count	Count the elements in an object
CurrentPath	Output the current path
Date	Output today's date
Define	Define a procedure
Define?, DefineP	Query existence of a procedure
Difference	Subtract two or more numbers
Dir	List the current directory
Dot	Draw a dot on the screen
DrawImage	Draw an image
Empty?, EmptyP	Test if an object has no elements
Eof?, EofP	Test for end-of-file
Equal?, EqualP	Test if two objects are equal
Exp	Exponential
ExportEPS	Export an EPS
ExportPDF	Export a PDF

Befehl	Funktion
ExportTIFF	Export a TIFF
Fill	Fill an area
FillIn	Fill an area
FillCurrentPath	Fill the current path
FillPath	Fill a path
First	Output the first element
FirstPut	Add an object to the start of another object
FontFace, Font	Return the name of the current font
FontFaces, Fonts	Return the names of available fonts
FontFamilies	Return the names of available font families
FontFamily	Return the name of the family of the current font
FontTraits	Return the traits of the current font
Forward, FD	Move the turtle forward
FPrint	Print to a file
FReadChar	Read a character from a file
FReadChars	Read characters from a file
FReadList	Read a line from a file into a list
FReadWord	Read a line from a file into a word
FShow	Write to a file
FType	Write to a file
GetMouseChange	Wait for the mouse button or a mouse move
GetMouseClick	Wait for the left mouse button to be pressed
GetMouseMoved	Wait for the mouse to be moved
GetProp, GProp	Retrieve a property for a name
GraphicsType, GrType	Draw Some Text
Heading	Output the turtle heading
HideTurtle, HT	Hide the turtle
Home	Home the turtle
If	Conditional processing
Instruments	Output available instruments
Integer	Truncate to integer
Item	Output nth element of an object
Last	Output the last element of an object
LastPut	Append to a word or list
Left	Turn the turtle anticlockwise
List	Create a list
List?, ListP	Test if object is a list
Local	Declare a local variable
Log, LN	Natural logarithm
Log10	Base-10 logarithm
LowerCase	Convert to lowercase
Make	Set the value of a variable
Member?, MemberP	Test membership of a word or list
Mouse	Output mouse co-ordinates
Name?, NameP	Test existence of a variable
Not	Logical NOT
Number?, NumberP	Test whether an object is a number
OpenAppend	Open a file for appending to
OpenRead	Open a file for reading
OpenTextAppend	Open a text file for appending to

Befehl	Funktion
OpenTextRead	Open a text file for reading
OpenTextWrite	Open a unicode file for writing to
OpenWrite	Open a file for writing to
Or	Logical OR
Output, Op	Output result from a procedure
PathBounds	Output the bounding box of a path
PathLength	Output the length of a path
Pen	Output the pen state and colour
PenColour, PenColor, PC	Output the pen colour number
PenDown, PD	Put the pen into drawing state
PenUp, PU	Put the pen into non-drawing state
PenWidth	Output the size of the pen
Pi	π
Play	Play sounds
Position, Pos	Output the turtle's position
Power	Raise a number to a power
Print	Display objects
Product	Multiplication
PropList, PList	List properties for a name
PutProp, PProp	Set a property for a name
Pwd	Output current directory
Quotient	Division
Random	Random number
ReadChar	Read a single character
ReadChars	Read multiple characters
ReadList	Read characters into a list
ReadWord	Read characters into a word
Remainder	Remainder from division
RemProp	Remove a property
Repeat	Repeat a list of statements
ReversePath	Reverse a path
RGB	Output the RGB values for a colour number
Right	Turn the turtle clockwise
Round	Round to nearest integer
Run	Execute a list of statements
Say	Speak using the system voice synthesizer
Sentence	Create a list
SetBackground, SetBG	Set the background colour
SetCanvasSize	Set the window size
SetClipPath	Set the clipping path
SetFontFace, SetFont	Set the current font face
SetFontFamily	Set the current font family
SetFontTraits	Set the traits for the current font
SetFullScreen	Enable fullscreen mode
SetHeading	Set the turtle's heading
SetLineCap	Set the ending style for lines
SetLineDash	Set the dash pattern for lines
SetPen	Set the state of the pen
SetPenColour, SetPenColor, SetPC	Set the colour for drawing

Befehl	Funktion
SetPenWidth	Set the width of the drawing pen
SetPosition, SetPos	Set the position of the turtle
SetRGB	Set a colour's RGB values
SetShadow	Set the dropshadow for drawing
SetTypeSize	Set the size for type
SetVoice	Set the current voice
SetWrap	Set the wrap state of the turtle
SetX	Set the x position of the turtle
SetY	Set the y position of the turtle
Shadow	Display the shadow state
Show	Display objects
Shown?	Output the visibility of the turtle
ShowTurtle, ST	Show the turtle
Sine, Sin	Sine
Sinh	Hyperbolic Sine
Snap	Capture an animation frame
SqRt	Square Root
Stop	Return from a procedure
StrokeCurrentPath	Stroke the current path
StrokePath	Stroke a path
Sum	Addition
Tangent, Tan	Tangent
Tanh	Hyperbolic Tangent
Text	Output a procedure as a list
TextBox	Output list describing text size
Thing	Output the value of a variable
Throw	Throw to a corresponding Catch
Time	Output the current time
Towards	Output required heading
Type	Print object
UpperCase	Convert to upper case
Voice	Output the name of the current voice
Voices	Output the names of available voices
Wait	Wait for a specified duration
WaitForSpeech	Wait for speech to finish
Word	Concatenate words
Word?, WordP	Test if object is a word
Wrap	Test if wrap is turned on
XPos	Output the turtle's x co-ordinate 54
YPos	Output the turtle's y co-ordinate