Turtlegrafik in Logo

Referent: Daniel Gent (deg@tzi.de)

Gliederung

- **Programmiersprache Logo**
  - **Übersicht**
  - **Interpreter**
  - **Variablen**
  - **Prozeduren**
  - **Kontrollstrukturen**
- **Verwendung der Turtle**
  - **Übersicht**
  - **Kontrolle der Turtle**
  - **Abfrage der Turtle**
- **Rekursives Zeichnen**
  - **Sierpinski-Dreiecke**
  - **Koch Kurve / Koch Schneeflocke**
- **Implementierung von L-Systemen**
  - **Sierpinski-Teppich**
  - **Busch3**

Logo

Logo - Übersicht

- entwickelt 1967 von Wally Feurzeig und Seymour Papert
- geplant als Einführung ins Programmieren für Kinder
- funktionale Programmiersprache
- interpretierte Programmiersprache

Logo – Interpreter

- Vielzahl von Interpretern für unterschiedliche Betriebssysteme
- für diesen Vortrag verwendeter Interpreter: UCBLogo (Berkley Logo), Version 5.5 für Windows

Logo – UCBLogo

- entwickelt von Brian Harvey und Studenten
- freie Software, unter GNU GPL vertrieben
- Versionen für Windows, Unix/Linux, MacOS X
- http://http.cs.berkeley.edu/~bh/

Logo - Sprache

- Unterscheidung zwischen Variablen und Prozeduren
  - Variable: Container für einen Wert
    - anzusprechen durch "NAME
  - Prozedur: Menge von Anweisungen
    - anzusprechen durch NAME
  - NAME darf gleichzeitig für Variable und Prozedur stehen

Logo - Variablen

- Drei verschiedene Datenstrukturen
  - Wörter
    - Hund, Katze, Maus, 1, 2, 3
  - Listen
    - Tiere = [Hund Katze Maus]
    - Liste von Wörtern, Listen oder Arrays
  - Arrays
    - Tiere = {Hund Katze Maus}@0

Logo - Variablen

- Typ wird dynamisch ermittelt

- Zwei Operationen möglich
  - Wert zuweisen
  - geschieht über die Prozedur **MAKE**
  - Beispiel: Zuweisung von 42 an die Variable Sinn
    - MAKE "SINN 42

Logo - Variablen

- Typ wird dynamisch ermittelt

- Zwei Operationen möglich
  - Wert ermitteln
  - geschieht über die Prozedur **THING**
  - Beispiel: Wert von Sinn ermitteln
    - **THING** *"SINN*
    - Syntactic Sugar
      - **:***SINN* equivalent zu **THING** *"SINN*

Logo - Prozeduren

- Menge von Anweisungen

- Beispiel

```
TO proc1
  PRINT "foo
END
```

Aufruf: proc1

Ausgabe: foo

Logo - Prozeduren

- Übergabe Parameter

- Beispiel

```
TO proc2 :p1
  PRINT :p1
END
```

Aufruf: proc2 "foo

Ausgabe: foo

Logo - Prozeduren

- Übergabe Parameter

- Beispiel

```
TO proc3 :p1
  PRINT "p1
END
```

Aufruf: proc3 "foo

Ausgabe: p1

Logo – Hello World

```
MAKE "HELLO [Hello World]

TO HELLOWORLD
 PRINT :HELLO
END
```

Aufruf: HELLOWORLD

Ausgabe: Hello World

Logo - Kontrollstrukturen

- Bekannte Kontrollstrukturen aus anderen Programmiersprachen verfügbar
  - Bedingte Anweisungen (IF, IFELSE, ...)
  - Schleifen (FOR, WHILE, ...)
  - ...

Logo - Kontrollstrukturen

- Beispiel: IF
  - IF tf instructionlist

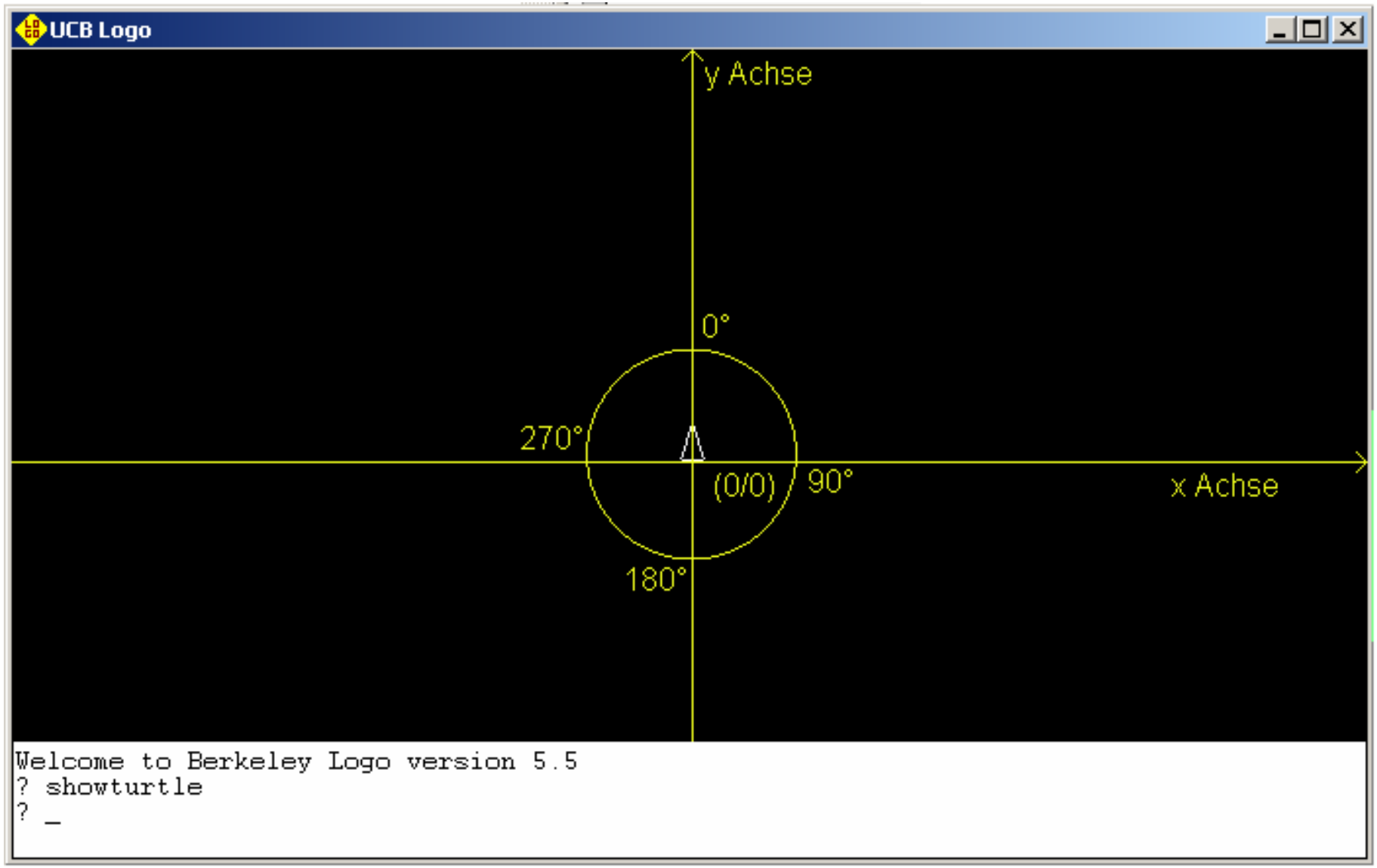
```
TO iftest :p1
 if :p1 > 0 [PRINT [> 0]]
 if :p1 < 0 [PRINT [< 0]]
 if :p1 = 0 [PRINT [=0]]
END
```

Turtle

Turtle - Übersicht

- befindet sich auf einer Ebene
- hat eine Position (x y) und eine Richtung (Grad relativ zur positiven y Achse im Uhrzeigersinn gemessen)
- kann sich bewegen
- kann bei der Bewegung eine Linie zeichnen

Turtle - Übersicht



Turtle - Kontrolle

- kann durch Kommandos manipuliert werden

- Beispiel
  - **FD** *100*
    - Turtle bewegt sich um 100 "Turtleeinheiten" in die Richtung in die sie schaut
  - **LT** *90*
    - Turtle dreht sich um 90° nach links

Turtle - Kontrolle

- nützliche Kommandos

FD *te*: um te in Blickrichtung laufen
LT *deg*: um deg nach links drehen
RT *deg*: um deg nach rechts drehen
SETHEADING *deg*: Richtung auf deg ändern
SETPOS (*xpos ypos*): Position auf (xpos ypos ändern)
PENUP: bei nachfolgenden Bewegungen keine Linie zeichnen
PENDOWN: bei nachfolgenden Bewegung Linie zeichnen

Turtle - Abfrage

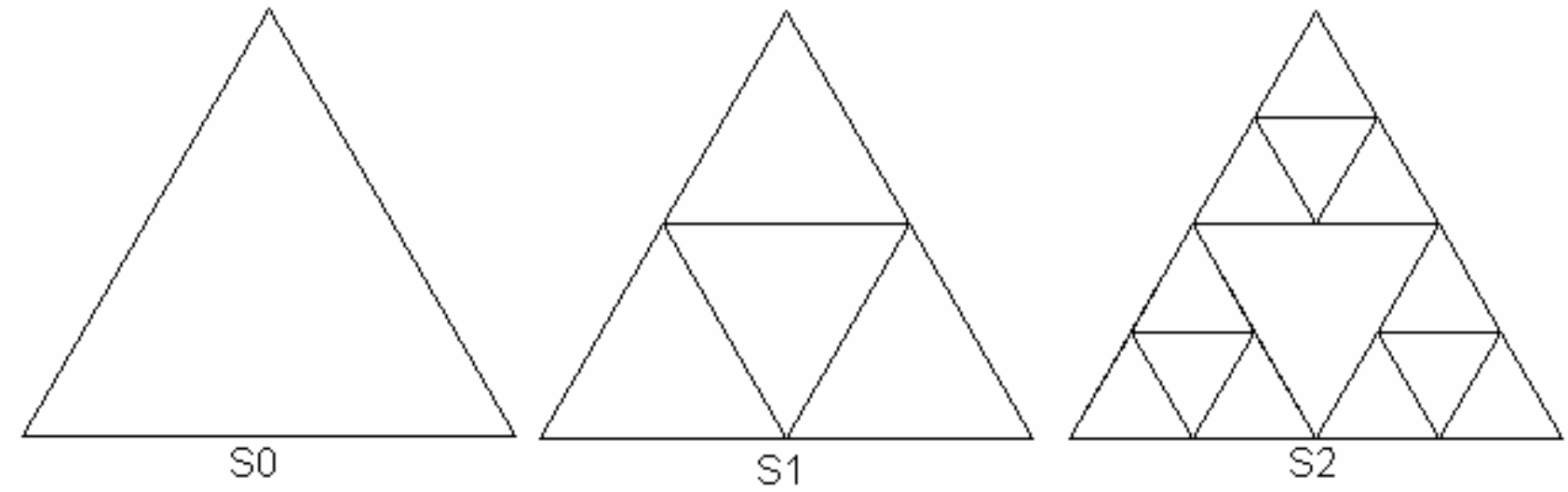
- die aktuellen Eigenschaften der Turtle können durch Kommandos abgefragt werden

- Beispiel
  - **POS**
    - liefert die Position der Turtle als eine Liste von Koordinaten
  - **HEADING**
    - liefert die Richtung der Turtle

Rekursives Zeichnen

Zeichnen – Sierpinski-Dreiecke

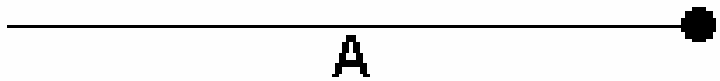
- Beispiel 1: **Sierpinski-Dreiecke**



Zeichnen – Sierpinski-Dreiecke

- Bauanleitung S0:

  A: Gehe um */* in Richtung *90* (rechts)

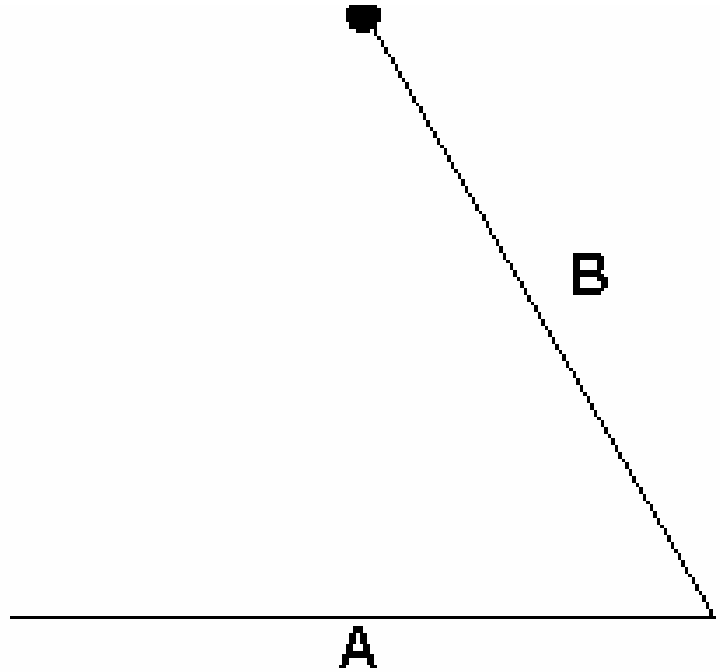


Zeichnen – Sierpinski-Dreiecke

- Bauanleitung S0:

  A: Gehe um *l* in Richtung *90* (rechts)

  B: Gehe um *l* in Richtung *330* (links oben)



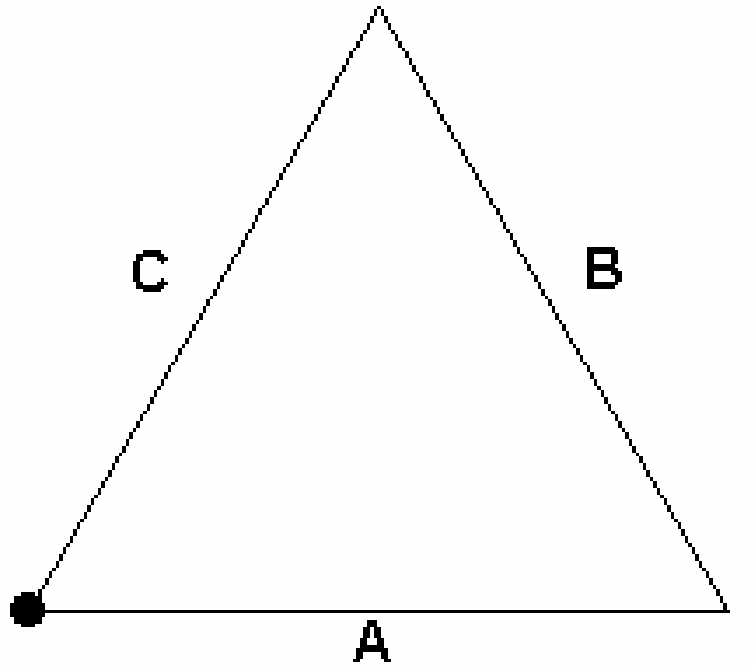
Zeichnen – Sierpinski-Dreiecke

- Bauanleitung S0:

  A: Gehe um *l* in Richtung *90* (rechts)

  B: Gehe um *l* in Richtung *330* (links oben)

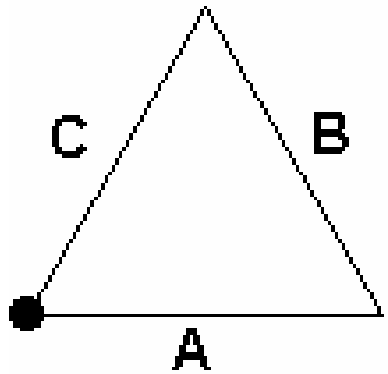
  C: Gehe um *l* in Richtung *210* (links unten)



Zeichnen – Sierpinski-Dreiecke

- Bauanleitung Sn (n>0):

  1: Zeichne S(n-1)

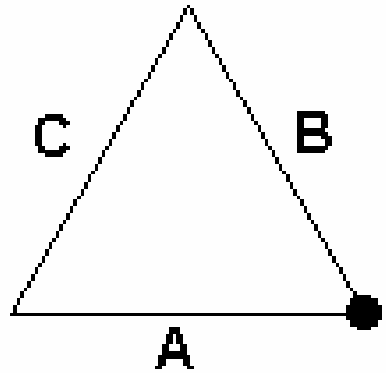


Zeichnen – Sierpinski-Dreiecke

- Bauanleitung Sn (n>0):

  1: Zeichne S(n-1)

  2: **A**



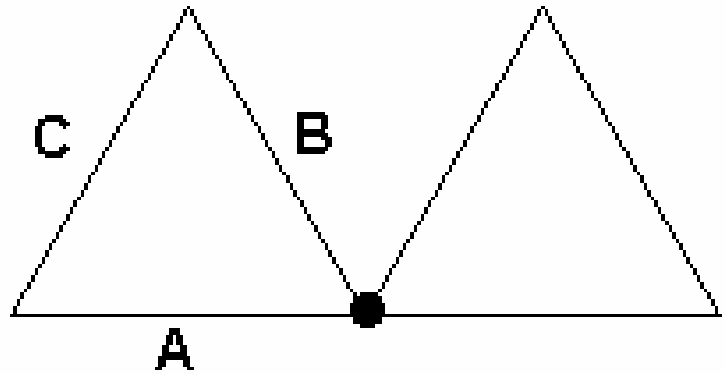
Zeichnen – Sierpinski-Dreiecke

- Bauanleitung Sn (n>0):

  1: Zeichne S(n-1)

  2: **A**

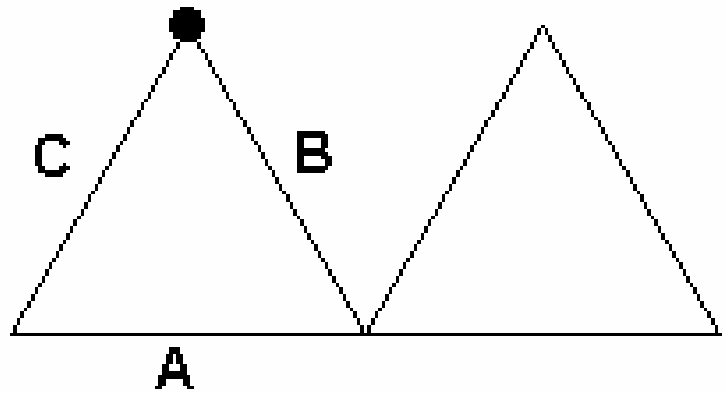
  3: Zeichne S(n-1)



Zeichnen – Sierpinski-Dreiecke

- Bauanleitung Sn (n>0):

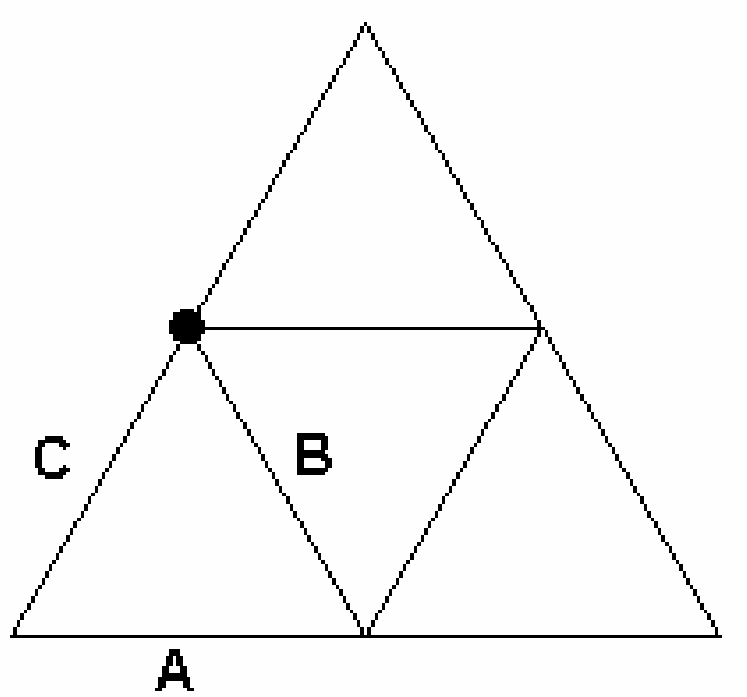
  1: Zeichne S(n-1)
  2: **A**
  3: Zeichne S(n-1)
  4: **B**



Zeichnen – Sierpinski-Dreiecke

- Bauanleitung Sn (n>0):

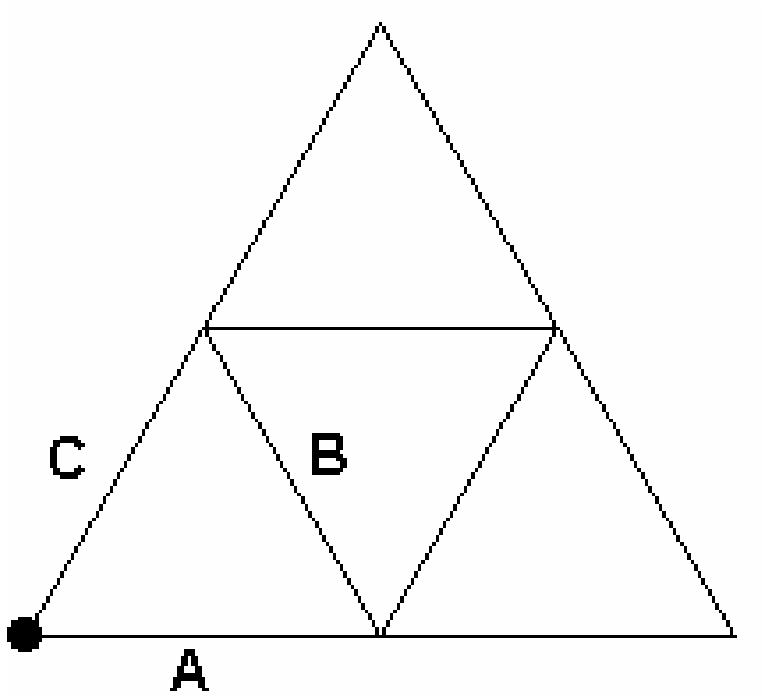
  1: Zeichne S(n-1)
  2: **A**
  3: Zeichne S(n-1)
  4: **B**
  5: Zeichne S(n-1)



Zeichnen – Sierpinski-Dreiecke

- Bauanleitung Sn (n>0):

  1: Zeichne S(n-1)
  2: **A**
  3: Zeichne S(n-1)
  4: **B**
  5: Zeichne S(n-1)
  6: **C**

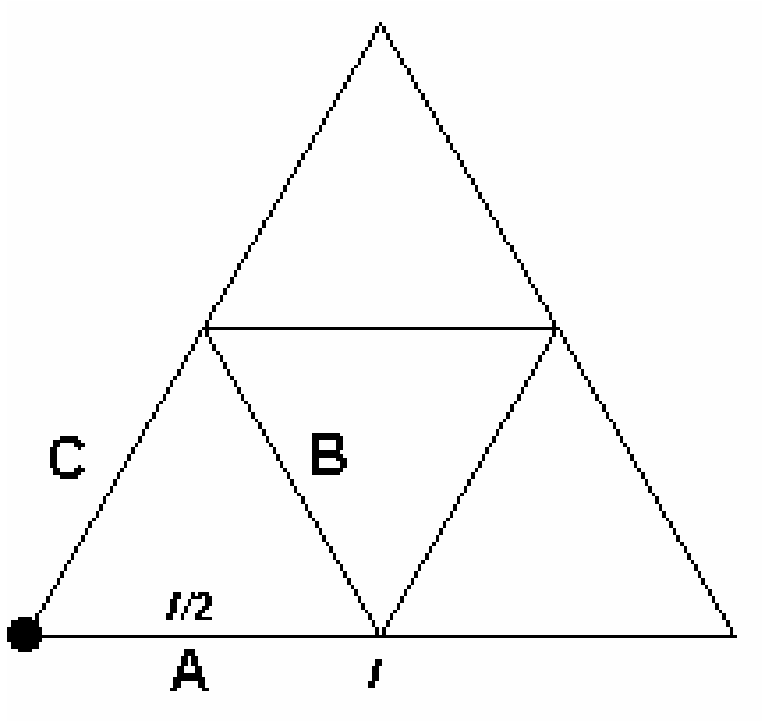


Zeichnen – Sierpinski-Dreiecke

- Bauanleitung Sn (n>0):

1: Zeichne S(n-1)
2: **A**
3: Zeichne S(n-1)
4: **B**
5: Zeichne S(n-1)
6: **C**

Skalierung beachten!



Algorithmische Umsetzung SD

1. Vorarbeit

```
;l = Länge der Dreieckseiten
;s = Stufe des zu erzeugenden Sirpinski-Dreiecks
TO sd :s :l
  CS ;Bildschirm löschen
  sdrec :s :l ;Rekursion starten
END
```

Algorithmische Umsetzung SD

2. A, B und C definieren

```
TO machA :l
  SETHEADING 90 ;nach rechts schauen
  FD :l ;um l in Blickrichtung gehen
END
```

Algorithmische Umsetzung SD

2. A, B und C definieren

```
TO machB :l
  SETHEADING 330 ;nach links oben schauen
  FD :l ;um l in Blickrichtung gehen
END
```

Algorithmische Umsetzung SD

2. A, B und C definieren

```
TO machC :l
  SETHEADING 210 ;nach rechts unten schauen
  FD :l ;um l in Blickrichtung gehen
END
```

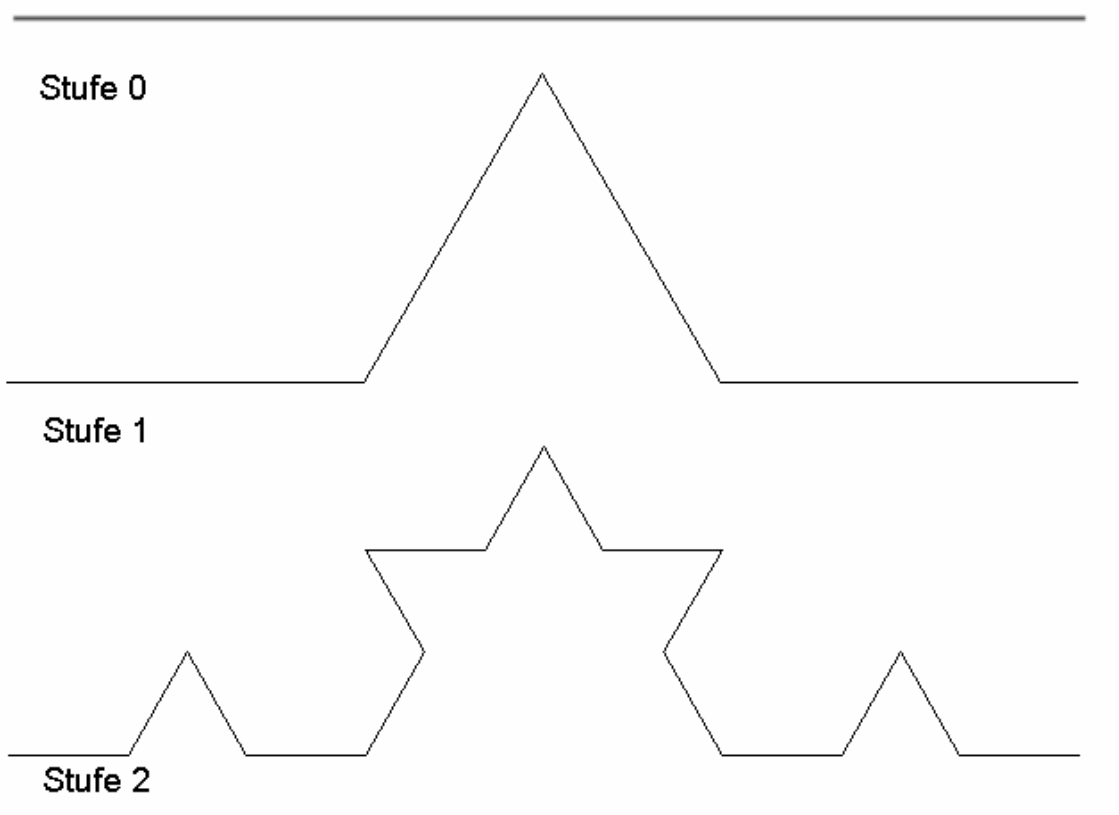
Algorithmische Umsetzung SD

3. Rekursion

```
TO sdrec :s :l
  IF :s = 0 [machA :l machB :l machC :l]
  IF :s > 0 [      sdrec :s-1 :l/2 machA :l/2
                   sdrec :s-1 :l/2 machB :l/2
                   sdrec :s-1 :l/2 machC :l/2]
END
```

Zeichnen – Koch-Kurve

- Beispiel 2: **Koch-Kurve**



Algorithmische Umsetzung KK

1. Vorbereitung

```
TO kk :s :l
  CS
  RT 90
  kkrect :s :l
END
```

Algorithmische Umsetzung KK

2. Rekursion

```
TO kkrec :s :l
  IF :s = 0 [FD :l]
  IF :s > 0 [     kkrec :s-1 :l/3 LT 60
                  kkrec :s-1 :l/3 RT 120
                  kkrec :s-1 :l/3 LT 60]
END
```

L-Systeme

L-Systeme

- **L-System**
  - **G = (V, s, P)**
    - **V: Alphabet (F, f, +, -, [,])**
    - **s: Axiom (Startwort)**
    - **P: Ableitungsregeln**

- **Annahmen**
  - **P kontextfrei**
  - **für jedes Zeichen des Alphabets genau eine Ableitungsregel**
    - **D0L-Systeme**

L-Systeme

- **Definition der Zeichen**
  - **F: Bewegung in Blickrichtung mit Linie**
  - **f: Bewegung in Blickrichtung ohne Linie**
  - **+: Drehung nach links**
  - **- : Drehung nach rechts**
  - **[: Position und Richtung der Turtle auf den Stack**
  - **] : Position und Richtung der Turtle vom Stack lesen und der Turle zuweisen**

L-Systeme – Implementierung

- **Definition des Stacks**
  - **Initialisieren**

```
MAKE "stack [ ]
```

  - **Pusch**

```
TO Pusch
  PUSH "stack LIST POS HEADING
END
```

  - **Popp**

```
TO Popp
  MAKE "tmp POP "stack
  SETPOS FIRST :tmp
  SETHEADING LAST :tmp
END
```

L-Systeme - Implementierung

- **Definition F f**
  - **F**

```
TO machFF :l
  FD :l
END
```

  - **f**

```
TO machf :l
  PENUP
  FD :l
  PENDOWN
END
```

L-Systeme - Implementierung

- **Definition + -**
  - **+**

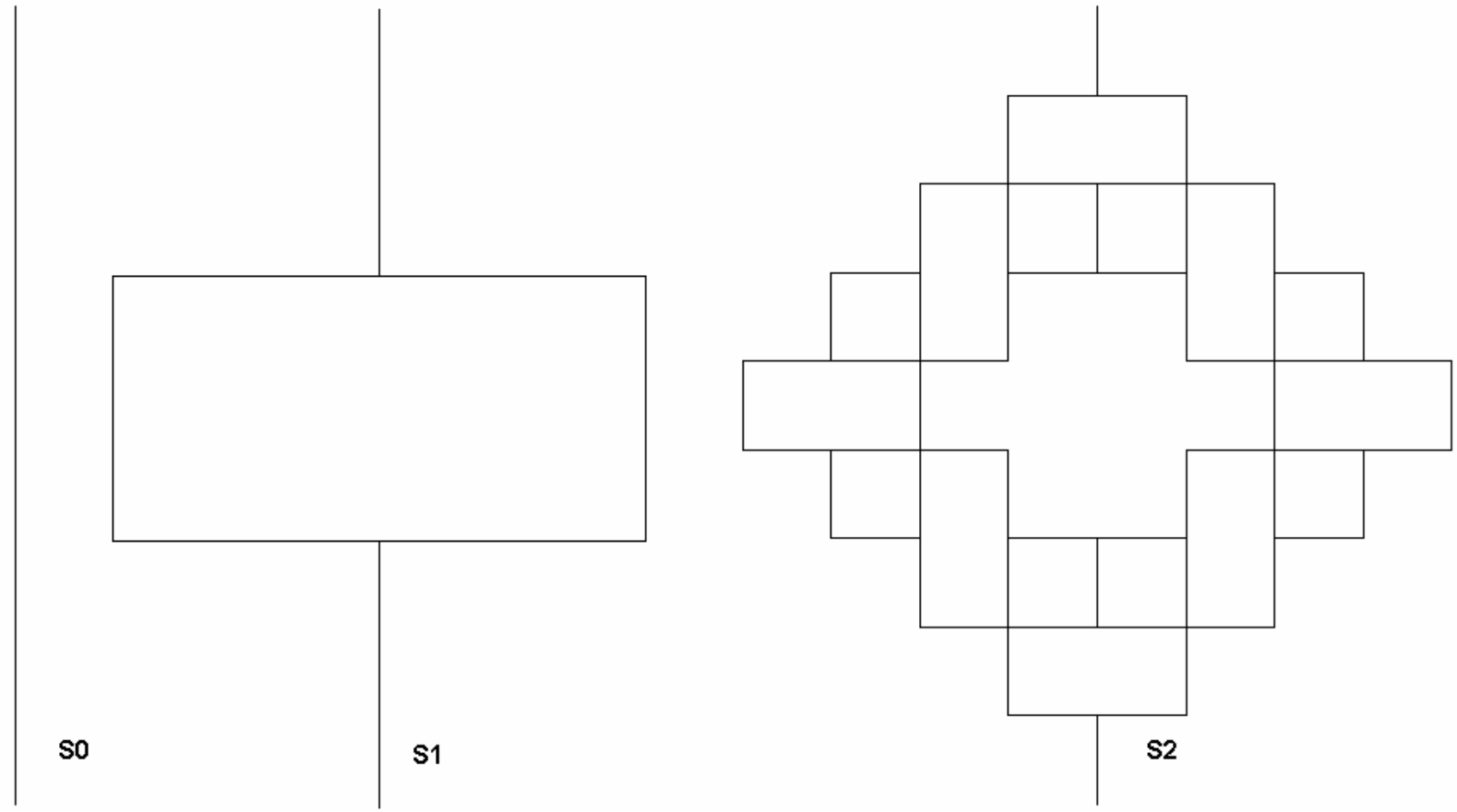
```
TO machPLUS :w
  TL :w
END
```

  - **-**

```
TO machMINUS :w
  TR :w
END
```

L-Systeme - Implementierung

- **Beispiel 1: Sierpinski-Teppich**



L-Systeme - Implementierung

- **Beispiel 1: Sierpinski-Teppich**
  - **Axiom: F**
  - **Ableitungsregeln**
    - **F -> F + F - F - F - f + F + F + F + F**
    - **f -> f f f**
  - **Sonstige Definitonen: Winkel 90°, Skalierung 1/3**

L-Systeme - Implementierung

- **Ableitungen definieren**

  F -> ...

```
TO stFF :s :l :w
  if :s = 0 [machFF :l]
  if :s > 0 [stFF :s-1 :l/3 :w
                machPlus :w
                ...
                stf :s-1 :l/3 :w
                ...]
END
```

L-Systeme - Implementierung

- **Ableitungen definieren**

  f -> f f f

```
TO stf :s :l :w
  if :s = 0 [machf :l]
  if :s > 0 [      stf :s-1 :l/3 :w
                   stf :s-1 :l/3 :w
                   stf :s-1 :l/3 :w]
END
```

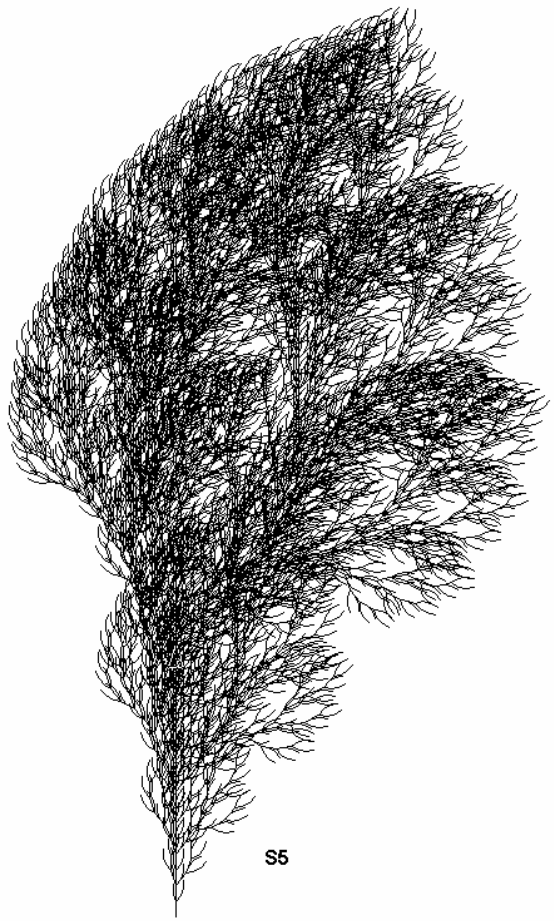
L-Systeme - Implementierung

- **Startprozedur**

```
TO stp :s :l
  CS
   stFF :s :l 90
 END
```

L-Systeme - Implementierung

- **Beispiel 2: Busch3**



L-Systeme - Implementierung

- **Beispiel 2: Busch3**
  - **Axiom: F**
  - **Ableitungsregeln**
    - **F -> F F - [- F + F + F] + [+ F - F - F]**
  - **Sonstige Definitonen: Winkel 20°, Skalierung ~1/2**

ENDE