Rekursion

VERWENDEN SIE FÜR DIE LÖSUNG Der AUFGABENTEILE A- C DIE VERZEICHNISSE UND SCRIPTE AUS DEM MITGELIEFERTEN PROJEKT A6 SoSe2015 Rekursion

A Rekursive Definitionen in ein Ruby-Programm übersetzen

1. Gegeben die folgende rekursive Definition zur Berechnung des Näherungswertes von ¼.

$$f(n) = \begin{cases} \frac{1}{6}; für\ n = 1 \\ \frac{1}{n * (n+1) * (n+2)} + f(n-1);\ für\ n > 1 \end{cases}$$

   a. Schreiben Sie eine Ruby-Methode für diese rekursive Definition.
   b. Testen Sie ihre Lösung. Ab welchem n wird der Fehler, die Abweichung der Näherung von dem exakten Wert 0.25, < 0.000001?

2. Gegeben die folgende rekursive Definition zur Berechnung des Näherungswertes von ln(x) für x > 0:

$$\ln(x,n) = \begin{cases} 2 * \frac{(x-1)}{(x+1)}\ ; für\ n = 0, x > 0 \\ 2 * \frac{(x-1)^{(2*n+1)}}{(2*n+1) * (x+1)^{(2*n+1)}} + \ln(x, n-1)\ ; für\ n > 0, x > 0 \end{cases}$$

   a. Schreiben Sie eine Ruby-Methode für diese rekursive Definition.
   b. Testen Sie ihre Lösung. Ab welchem n wird der Fehler, die Abweichung der Näherung von dem exakten Wert ln(2) < 1e-15. Verwenden Sie zur Berechnung des exakten Wertes die Methode ***Math.log***.

3. Ackermann Funktion
   Die Ackermann Funktion ist eine extrem schnell wachsende rekursive Funktion. Sie wird in der Praxis z.B. verwendet, um die Leistungsfähigkeit von Programmiersprachen zu testen. Es gibt verschiedene Varianten der Definition der Ackermann-Funktion. Wir verwenden die einfachere Variante von Rósza Péter:

$$a(n,m) = \begin{cases} m+1; für\ n = 0 \\ a(n-1,1); für\ m = 0 \\ a\big(n-1, a(n,m-1)\big); für\ n > 0, m > 0 \end{cases}$$

   a. Schreiben Sie eine Ruby-Methode für diese rekursive Definition.
   b. Für welches m berechnet Ihre Lösung a(3,m) noch?
   c. Rufen Sie die Funktion für die nachstehenden Kombinationen von n/m auf. Was beobachten Sie?

n / m	0	1	2	3	4	m
0	1	2	3	4	5	m+1
1	2	3	4	5	6	m+2
2	3	5	7	9	11	2*m+3
3	5	13	29	61	125	$8*2^m-3$
4	13	65333				

B Rekursive Programme in endrekursive Programme überführen

1. Formen Sie die Methoden aus A-1 und A-2 in endrekursive Lösungen um.
2. Zeigen Sie, dass die Umformungen korrekt arbeiten, in dem Sie für mehrere n in 1..100, die Ergebnisse der Methoden aus A mit denen aus B vergleichen.
   Nachfolgende Ausgaben sollte der Vergleich erzeugen:

```
f(1)-f_sp(1): 0.0
ln(2,1)-ln_sp(2,1): 0.0
f(2)-f_sp(2): 0.0
ln(2,2)-ln_sp(2,2): 0.0
f(3)-f_sp(3): 0.0
ln(2,3)-ln_sp(2,3): 0.0
f(4)-f_sp(4): -2.7755575615628914e-17
ln(2,4)-ln_sp(2,4): 0.0
...
f(99)-f_sp(99): 0.0
ln(2,99)-ln_sp(2,99): -2.220446049250313e-16
f(100)-f_sp(100): 2.7755575615628914e-17
ln(2,100)-ln_sp(2,100): -2.220446049250313e-16
```

C Iterative Formeln in rekursive Ruby-Programme übersetzen

1. Gegeben die iterative Näherungsformel für ln(x) für x > ½:

$$\sum_{i=1}^{n} \frac{(x-1)^i}{i * x^i}; für\ x > 0.5$$

   a. Formen Sie die iterative Definition in eine rekursive um.
   b. Schreiben Sie eine Ruby-Methode, die die Formel rekursiv berechnet.

2. Gegeben die iterative Näherungsformel für $1/(1+x)^2$ für |x| <1:

$$\sum_{i=0}^{n} (-1)^i * (i+1) * x^i$$

   a. Formen Sie die iterative Definition in eine rekursive um.
   b. Schreiben Sie eine Ruby-Methode, die die Formel rekursiv berechnet.

D Lindenmayer Systeme und rekursive Grafiken

Pflanzen aber auch andere Naturphänomene lassen sich als Fraktale begreifen. Der Begriff Fraktal geht zurück auf den Mathematikern Benoît Mandelbrot und beschreibt natürliche oder künstliche Muster mit einem hohen Grad an Selbstähnlichkeit. Fraktale entstehen z.B. wenn sich Objekte aus mehreren verkleinerten Kopien zusammensetzen lassen. Fraktale sind daher ein weiterer Vertreter für rekursive Strukturen.

Der Biologe Aristid Lindenmayer entwickelte unter Verwendung von Chomsky Grammatiken einen Formalismus, mit dem sich natürliche Muster mit Hilfe von Grammatiken beschreiben lassen. Dieser Formalismus, genannt Lindenmayer Systeme (kurz L-Systeme) wurde zuerst für die Berechnung primitiver multizellulärer Organismen verwendet. In Zusammenarbeit mit dem Mathematiker Przemyslaw Prusinkiewicz gelang es die mathematischen Grammatiken mit Hilfe von Turtle-Grafiken zu visualisieren. Heutzutage sind L-Systeme soweit, dass sich sehr realistische Darstellungen natürlicher Formen, Pflanzen, Gebirge etc. im Computer erzeugen lassen (vgl. Abbildung 1) [1].

Abbildung 1: Computergenerierter Baum (http://michael.szell.net/fba/kapitel2.html#kapitel21) / Sonnenblume (http://michael.szell.net/fba/kapitel4.html) / Sträucher („Fractal weeds“. Lizenziert unter Gemeinfrei über Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Fractal_weeds.jpg#/media/File:Fractal_weeds.jpg)

Das Grundprinzip der L-Systeme

L-Systeme basieren auf der Ersetzung von Teilen eines Objektes mit Hilfe von Regeln. Objekte werden abstrakt als Zeichenketten beschrieben, in denen Zeichen / Symbole eine bestimmte Bedeutung zugeordnet werden. Formal lässt sich eine L-System als Tupel L=(V,w,P) beschreiben, wobei V die gültigen Zeichen (das Alphabet), w die Startsymbole und P die Produktionsregeln beschreibt.

Tabelle 1: Symbole und deren Bedeutung in L-Systemen

Symbol	Bedeutung
F	Vorwärtsbewegung und Zeichnen
f	Vorwärtsbewegung ohne Zeichnen
+	Linksdrehung
-	Rechtsdrehung

Durch eine geeignete grafische Interpretation der Symbole, können L-Systeme mit Hilfe von Turtle-Grafik visualisiert werden.

Erstes kleines Beispiel

L-System für die Kochkurve

G = (V,w,P)

V= {F,+,-} # die Symbole

w= {F } # das Startsymbol

P= { F-> F+F--F+F }, # Regel für die Kochkurve

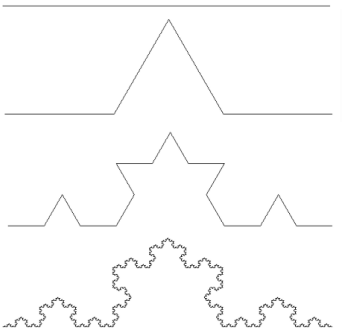

Abbildung 2: Kochkurven der Ordnung 0,1,2 und 5. Erzeugt mit der Turtlegrafik der Toolbox. Vorlage siehe [1].

Bei der Kochkurve wird eine Strecke jeweils durch 4 verkürzte Strecken ersetzt. Wenn der Drehwinkel 60° ist, dann ergeben sich geometrisch die Längen der Teilstrecken als Drittel der ursprünglichen Strecke.

Eine Kochkurve 0'ter Ordnung ergibt sich, wenn keine Ersetzung von F erfolgt.

F

Eine Kochkurve 1'ter Ordnung ergibt sich, wenn 1 Ersetzung von F über Regeln erfolgt.

F+F--F+F

Eine Kochkurve 2'ter Ordnung ergibt sich, wenn 2 Ersetzungen von F über Regeln erfolgen, u.s.w.

F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F

Turtle Grafik

Turtle Grafik, erfunden von Seymour Papert und implementiert in der Sprache LOGO, basiert auf der Idee, dass sich beliebige geometrische Formen im 2D Raum durch Aneinandersetzen von Strecken erzeugen lassen. Eine Turtle (Schildkröte) bewegt sich über das Zeichenblatt und hinterlässt eine Spur, wenn der „Schwanz" unten ist (Symbol **F** in Tabelle 2, Symbol **f** steht für Bewegungen ohne zu zeichnen („Schwanz ist oben")). Die Turtle kann sich in beliebige Richtungen drehen. Dabei werden Drehungen gegen den Uhrzeigersinn auf Linksdrehungen (Symbol **+** in Tabelle 2), Drehungen im Uhrzeigersinn als Rechtsdrehungen (Symbol **–** in Tabelle 2) dargestellt.

Dies ist die einfachste Variante der Turtle-Grafik mit der sich bereits bestimmte Kategorien von L-Systemen darstellen lassen.

Tabelle 2: **Symbole und deren Interpretation in der Turtle-Grafik**:

Symbol	**Bedeutung**	**Methoden der Klasse *Turtle***
F(steps)	gehe steps Schritte voraus und zeichne eine Strich	***go_ahead***
f(steps)	gehe steps Schritte voraus ohne zu zeichnen	***go_head***
+(alpha)	drehe dich um alpha nach links	***turn_left***
-(alpha)	drehe dich um alpha nach rechts	***turn_right***

Programmierschnittstelle der Turtle-Grafik

Die Implementierung der Turtle-Grafik, die Klasse ***Turtle***, setzt die Interpretation aus Tabelle 2 in animierte Grafikbefehle um.

Die Schnittstelle der ***Turtle*** ist wie folgt definiert

`initialize(x,y,angle)`	Setzt ein ***Turtle***-Objekt auf Position (***x,y***) und richtet sie nach Winkel ***angle*** im Uhrzeigersinn aus. Wenn kein Wert für den Winkel übergeben wird, dann schaut die Turtle nach rechts (Osten). Der „Schwanz“ der Turtle ist zu Beginn unten.
`turn_left(angle)`	Bewege die ***Turtle*** um den Winkel ***angle*** **gegen** den Uhrzeigersinn. Entspricht dem "**+**" aus Tabelle 2
`turn_right(angle)`	Bewege die ***Turtle*** um den Winkel ***angle*** **im** den Uhrzeigersinn. Entspricht dem "**-**" aus Tabelle 2.
`go_ahead(steps,width)`	Gehe Anzahl Schritte ***steps*** in Richtung Blickrichtung. ***Width*** gibt die ***Breite*** der Spur vor, also die Breite des Zeichenstifts. Wird ***width*** nicht angegeben, dann ist die Stärke 1. Ob gezeichnet wird, hängt davon ab, ob der „Schwanz“ der Turtle oben oder unten ist. Dies wird über die Methode ***down*** und ***up*** gesteuert. Entspricht dem **F/f** in Tabelle 2.
`up()`	Bewegt den „Schwanz“ der Turtle nach oben.
`down()`	Bewegt den „Schwanz“ der Turtle nach unten.

Kochkurve mit TurtleGrafik

Für das L-System der Kochkurve müssen wir noch die *Startlänge*, den *Verkürzungsfaktor* und den *Drehwinkel* festlegen:

Startlänge l: möglichst nicht zu kurz, da in jedem Ersetzungsschritt verkürzt wird (z.B. 600)

Verkürzungsfaktor: 1/3

Drehwinkel: 60°

Dann können wir aus der rekursiven Definition des L-Systems und den Zusatzinformationen Startlänge, Verkürzungsfaktor und Drehwinkel eine rekursive Implementierung für die Kochkurve angeben.

```ruby
class KochKurve

  # wdhl gibt die Anzahl der rekursiven Verzweigungen und damit die Ordnung der
  # Kochkurve an
  # x,y ist die Anfangsposition der Turtle und
  # kl die Kantenlänge zu Beginn
  #
  # Der Drehwinkel ist eine für die Kochkurve feste Größe von 60 im Gradmass
  # @factor ist die Verkürzung in jedem Rekursionsschritt

  def zeichnen(wdhl,x,y,kl)
    @turtle = Turtle.new(x,y,0)
    @angle = 60
    @factor = 3.0
    koch(wdhl,kl)
  end

  # Löscht alle Zeichnungen der Turtle
  def loeschen()
    @turtle.loeschen()
  end

  # rekursive Funktion zum Zeichnen der Kochkurve
  def koch(n,kl)
    # Abbruchbedingung: Alle Ersetzungen wurden abgeschlossen
    # jetzt können die verkürzten Strecken gezeichnet werden
    if n == 0
      @turtle.go_ahead(kl)
      return
    end
    # Rekursiver Aufruf - Anwendung der Ersetzungsregel: F -> F+F--F+F
    koch(n-1,(kl/@factor).round)              # F
    @turtle.turn_left(@angle)                 # +
    koch(n-1,(kl/@factor).round)              # F
    @turtle.turn_right(@angle)                # -
    @turtle.turn_right(@angle)                # -
    koch(n-1,(kl/@factor).round)              # F
    @turtle.turn_left(@angle)                 # +
    koch(n-1,(kl/@factor).round)              # F
  end

end
```

Aufgabe D

Lèvy Fraktal:

Die geometrische Konstruktionsidee für das Lèvy Fraktal lautet: Zeichne eine Strecke der Länge kl. Errichte über dieser Strecke ein rechtwinkliges Dreieck mit Katheten gleicher Länge. Fahre mit der Konstruktionsvorschrift für die beiden Katheten fort. Zeichne nur die Katheten, nicht die Hypotenuse.

Ein Lévy-Fraktal der Ordnung 0 ist eine Strecke. Ein Lèvy-Fraktal der Ordnung 1 ist ein einfacher „Hut“. Damit ein Hut der Ordnung 1 entsteht, muss eine 45° Drehung nach links ausgeführt werden, dann ein Lévy-Fraktal der Ordnung 0 mit Streckenlänge $\frac{1}{\sqrt{2}}$ *kl gezeichnet werden, dann eine 90° Drehung nach rechts ausgeführt werden und schließlich wieder ein Lévy-Fraktal der Ordnung 0 mit Streckenlänge $\frac{1}{\sqrt{2}}$ *kl gezeichnet werden ($\frac{1}{\sqrt{2}}$ *kl ergibt aus dem Satz des Pythagoras).

Diese Vorschrift wird in der Produktionsregel des L-Systems ausgedrückt und lässt sich zur rekursiven Vorschrift für die Konstruktion von Levy-Fraktalen höherer Ordnung verstehen. Da in der Produktionsregel ein **F** durch **2 F** ersetzt wird und nur für Lèvy-Fraktale der Ordnung 0 eine Strecke gezeichnet wird, ist sichergestellt, dass nur die Katheten der kleinsten Dreiecke der Ordnung n gezeichnet werden. Der Verkürzungsfaktor $\frac{1}{\sqrt{2}}$ gibt das Verhältnis zwischen altem **F** und den **2 neuen F** wieder.

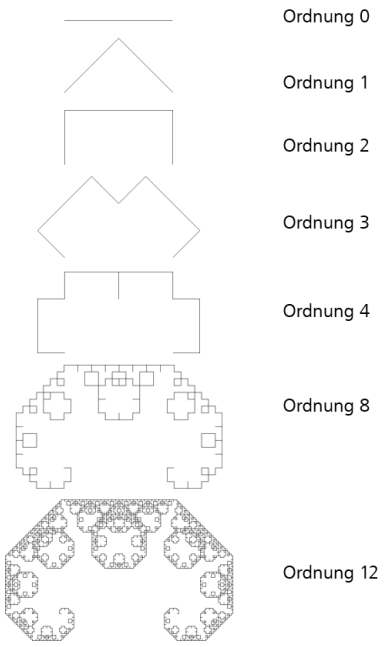

Abbildung 3: Lèvy-Fraktale verschiedener Ordnungen für Kantenlänge 300. Generiert mit der Turtle-Grafik der Toolbox.

L-System für das Lèvy-Fraktal

Definition nach [2]:

G = (V,w,P)
V = {F,+,-}
w = {F}
P = {F → +F--F+}

Drehwinkel: 45°
Verkürzungsfaktor: $\frac{1}{\sqrt{2}}$

Implementieren Sie im Projekt *A6 SoSe2015 RekursiveGrafik* in der Klasse ***LevyFraktal*** die Methode ***levy(n,kl)***, die die rekursive Produktionsregel mit Hilfe der Turtle grafisch umsetzen soll. Winkel und der Nenner des Verkürzungsfaktors werden bereits bei der Initialisierung des Fraktals definiert.

Hinweis:

- Weitere L-Systeme für Kurven und Fraktale finden Sie z.B. auf der Seite http://www-user.uni-bremen.de/schmuhl/fractals/fractal_curves.html
- Im Projekt *A6 SoSe2015 RekursiveGrafik* finden Sie für 4 weitere Fraktale die vorbereiteten Klassen. Wenn Sie Spaß an solchen Aufgaben haben, versuchen Sie sich an einer Lösung ☺.

E Lindenmayer Systeme und Bäume

Auch das Wachstum/Entstehen „natürlicher" Bäume lässt sich rekursiv auf Basis von Lindenmayer Systemen beschreiben. Ein Baum entsteht, indem an gewissen Knotenpunkten Verzweigungen angesetzt werden, die selbst wieder Bäume sind.

Da die Verzweigungen am gleichen Knotenpunkt ansetzen, muss es möglich sein, nach Fertigstellung eines Zweiges / Teilbaums, auf den Startknoten dieses Teilbaums zurückzusetzen. Dazu müssen wir die Symbole des L-Systems um zwei weitere ergänzen: eines, um uns den Zustand vor der Verzweigung zu merken (das Symbol **[**) und eines um den alten Zustand wiederherzustellen (das Symbol **]**). Die vollständige Liste der Symbole für diese Varianten eines L-Systems sowie die zugehörigen Methoden der Turtle-Grafik zeigt Tabelle 3.

Tabelle 3: **Symbole und deren Interpretation in der Turtle-Grafik**:

Symbol	**Bedeutung**	**Methode der Klasse *Turtle***
F(steps)	gehe steps Schritte voraus und zeichne eine Strich	***go_ahead***
f(steps)	gehe steps Schritte voraus ohne zu zeichnen	***go_head***
+(alpha)	drehe dich um alpha nach links	***turn_left***
-(alpha)	drehe dich um alpha nach rechts	***turn_right***
[	speichere den aktuellen Zustand	***remember***
]	Stelle den letzten Zustand wieder her	***restore***

Ein ***Turtle***-Objekt speichert beim Aufruf von ***remember*** als aktuellen Zustand die Position und die Blickrichtung auf einem Stack. Mit ***restore*** wird das oberste Element – der letzte gespeicherte Zustand – vom Stack gelesen und Position und Winkel der Turtle auf den letzten gespeicherten Zustand gesetzt. Durch die Verwendung eines Stacks ist es möglich auch geschachtelte Zustände (geschachtelte Klammerung) in den Regeln der L-Systeme zu verwenden.

Aufgabe E:

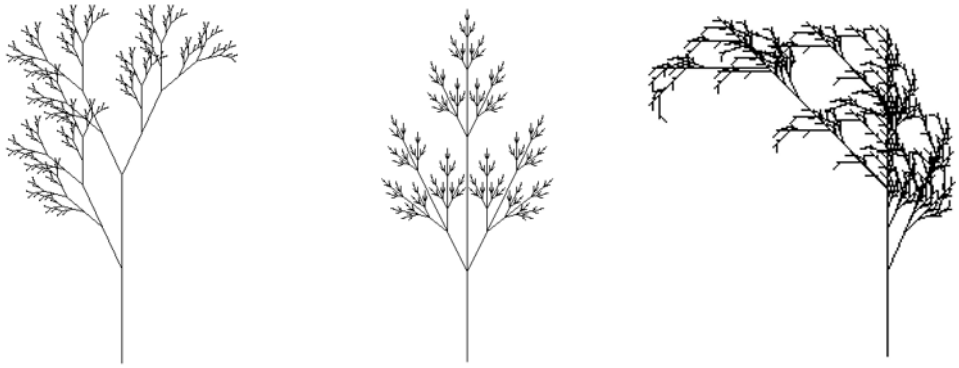

Abbildung 4: 3 L-Systeme mit Zustandsspeicher an Verzweigungspunkten – Baumtyp 1 der Ordnung 9 mit Länge=100, Breite konstant 1, δ=24.8; Baumtyp 2 der Ordnung 9 mit Länge=100, Breite konstant 1, δ=25.7; Baumtyp 3 der Ordnung 7 mit Länge=50, Breite konstant 1, δ=22.5. Generiert 2015 mit der Turtle-Grafik der Toolbox. Vorlagen siehe [1].

Abbildung 4 zeigt die Grafiken für 3 L-Systeme mit Zustandsspeicher an den Verzweigungspunkten mit unterschiedlichen Drehwinkeln, Streckenlängen sowie Produktionsregeln.

1. Übersetzen Sie die nachfolgenden L-Systeme in rekursive Methoden ***baum1***, ***baum2***, ***baum3*** der Klasse ***RekursiverBaum*** unter Verwendung der Turtle-Grafik.
2. Wie müssen Sie Lösungen modifizieren, damit die Äste des Baumes in jedem Schritt schmaler werden? Nehmen Sie für einen Baumtyp diese Modifikation vor.
3. Modifizieren Sie die Implementierung für einen Baumtyp derart, dass der rechte und linke Ast in unterschiedlichen Winkeln gezeichnet werden.

L-System für Baumtyp 1 (linke Grafik)

w = X
P = { X → F[+X]F[-X]+X }
Drehwinkel: 24.8°
Verkürzungsfaktor: 1/2

L-System für Baumtyp 2 (mittlere Grafik)

w = X
P = { X → F[+X][-X]FX }
Drehwinkel: 25.7°
Verkürzungsfaktor: ½

L-System für Baumtyp 3 (rechte Grafik)

w = X
P = { X → F-[[X]+X]+F+[+FX]-X }
Drehwinkel: 22.5°
Verkürzungsfaktor: ½

Quellen:

[1] L-Systeme: http://michael.szell.net/fba/kapitel2.html#kapitel21

[2] L-Systeme: http://www-user.uni-bremen.de/schmuhl/fractals/fractal_curves.html