Musterlösungen zu den Aufgaben aus dem Artikel Algorithmen der fraktalen Geometrie Teil 1: Chaos, Fraktale und Drachenkurve

Zitate aus dem Artikel sind in dieser Schrift gesetzt, *die Lösungsvorschläge kursiv.*

Die Zahl der möglichen Faltungen ist praktisch begrenzt: „A well-known empirical theorem states that a piece of paper cannot be folded in half more then seven times, no matter how long or thin a sheet is used“ (Davis/Knuth 1970, S. 67).

Aufgabe 1: Bestätigen oder widerlegen das zitierte empirische Theorem! Wie hängen Länge der Abschnitte und Zahl der Knicke von der Iterationstufe *n* ab?

In einer Folge der „Sendung mit der Maus“ wurde das Papierfalten zum Thema gemacht. Dort wurde der Bodybuilder Mirko aufgefordert, ein Papier mehr als siebenmal zu falten – das gelang ihm nicht! Also 1:0 für Davis/Knuth. In der Reaktion auf die Sendung ist es aber verschiedenen Kindern gelungen, diese Grenze zu durchbrechen, s. http://www.ardmediathek.de/ard/servlet/content/684138. Es zeigte sich, dass es mit dünnem Zeitungs- oder Seidenpapier möglich ist, ein Papier auch acht- oder neumal in der Mitte zu falten, also 1:2 gegen Davis/Knuth. Ähnliche Ergebnisse wurden seinerzeit als Leserzuschriften auch in der Zeitschrift veröffentlicht, die die Artikel von Davis/Knuth gebracht hatte. Wenn man andere Faltungsmethoden – wie in der „Sendung mit der Maus“ gezeigt – verwendet, sind noch mehr Faltungen möglich, allerdings entspricht das nicht mehr der Bedingung „folded in half“.

Die Länge der Abschnitte ergibt sich zu $L \cdot (1/2)^n$, wobei L die Länge des Papierstreifens und n die Iterationsstufe ist, die Zahl der Knicke beträgt dann $2^n - 1$.

Bei der Untersuchung der Ziffern an den ungeraden Stellen (die unterstrichenen Ziffern) ergibt sich nach Streichung der anderen Ziffern immer die ziemliche eintönige Folge 10101010.... Die nicht unterstrichenen Ziffern ergeben dagegen immer die Dualzahl aus der vorangegangen Iterationsstufe (Davis/Knuth 1970, S. 67, Albers 2006, Kapitel 3).

Aufgabe 2: Überlegen Sie sich eine anschauliche Begründung für die zuletzt genannte Eigenschaft.

Das in dem Artikel erklärte Gesetz mit der Spiegelung und Inversion wird von der Bremer Gruppe Reflexionsgesetz genannt. Die eben erwähnte Beobachtung nennen sie Inflationsgesetz, weil die vorherige Stufe mit der Folge 1010101010... „aufgeblasen“ wird. Eine anschauliche Erklärung ergibt sich, wenn man die Knicke der vorherigen Stufen auf dem Papierstreifen markiert, den Streifen nochmals faltet und nur die neu hinzugekommenen Knicke betrachtet – dann ergibt sich die erwähnte eintönige Folge: Der letzte Knick ergibt immer abwechselnd Links- und Rechts knicke, also 1 und 0 im Wechsel. In einer Präsentation von Reimund Albers wird dies sehr schön veranschaulicht, s. http://www.mevis-research.de/~albers/Materialien/neuesPapfalten.zip oder noch schöner für die Mac-User mit dem Programm Keynote http://www.mevis-research.de/~albers/Materialien/neuesPapfalten_key.zip.

Aufgabe 3: Schreiben Sie ein Programm, das bei Eingabe der Iterationsstufe die jeweilige Näherung der Papierfaltungszahl als Dualbruch ausgibt.

Aufgabe 4: Berechnen Sie die Papierfaltungszahl auf Taschenrechnergenauigkeit.

Für die Lösung der Aufgaben 3, 4, 8 und 9 kann das folgende Python-Programm verwendet werden.

```
# -*- coding: cp1252 -*-
# Musterlösung zu den Aufgaben 3, 4, 8 und 9 mit Python
# H. Witten, 09.12.2008

def neue_Stufe_refl(s):
    '''
    Berechnet die neue Stufe der Papierfaltungszahl
    nach dem Reflexionsgesetz.
    '''
    s_refl = ''     # Berechnung des reflektierten Strings
    for i in range(len(s)-1,-1,-1): # Indizes von s rückwärts
        if s[i] == '1': s_refl += '0'
        elif s[i] == '0': s_refl += '1'
    return s+'1'+s_refl     # Das Reflexionsgesetz

def neue_Stufe(s):
    '''
    Berechnet die neue Stufe der Papierfaltungszahl
    nach dem Inflationsgesetz.
    Beispiel:
     1 0 1 0 1 0 1 0 => s_infl (= 4*'10', 4 == (7+1)/2)
      1 1 0 1 1 0 0  => s      (len(s) == 7)
     110110011100100 => Rückgabestring von "neue_Stufe"
    '''
    s_infl = '10'*((len(s)+1)/2)
    s_neu = ''
    for i in range(len(s)):
        s_neu += s_infl[i]+s[i]
    return s_neu+'0'


def Papierfaltungszahl(n):
    '''
    Eingabe: Iterationsstufe n, Ausgabe: Papierfaltungszahl
    Lösung für Aufgabe 3, wenn vor dem Ausgabestring '0.'
    angehängt wird.
    '''
    if n == 0: return ''
    s = '1'
    for i in range(n-1):
        s = neue_Stufe(s)
    return s

def Naeherung(s):
    '''
    Eingabe: Papierfaltungszahl als String mit '0' und '1'
    Ausgabe: rationaler Näherungswert für die Papierfaltungszahl
    Berechnung mit dem Horner-Schema.
    Ab n == 5 ändert sich der ausgegebene Wert nicht mehr:
    >>> Naeherung(Papierfaltungszahl(5))
    0.85073618820186725
    Lösung für Aufgabe 4
    '''
    hilf = int(s[len(s)-1]) # Die letzte Ziffer von s
    for i in range(len(s)-2,-1,-1): # Indizes von s rückwärts
        if s[i] == '1': hilf = 1 + 0.5*hilf
        elif s[i] == '0': hilf = 0.5*hilf
    return 0.5*hilf
```

```
import turtle
from math import log

def TurtleInit(s):
    '''
    Initialisierung der Turtle
    Es empfiehlt sich, Python 2.6 oder bei älteren Versionen das
    xturtle-Modul von Gregor Lingl zu verwenden. Das Programm funktio-
    niert aber auch mit dem alten turtle-Modul (für Versionen < 2.6 im
    Lieferumfang von Python enthalten).
    '''
    global n                # unschön, aber bequem
                            #(wird nur für die Positionierung und
                            # Skalierung der Grafik verwendet)
    n = int(log(len(s)+1)/log(2)) # Iterationstufe aus len(s) bestimmen,
                            # zur Formel s. Aufgabe 1!
    turtle.up()             # Turtle passend ausrichten
    turtle.backward(200)
    turtle.right(n*45)
    turtle.down()           # ...jetzt kann's losgehen!!!

def Drachenmuster(s):
    '''
    Eingabe: Papierfaltungszahl als String mit '0' und '1'
    Ausgabe: die entsprechende Drachenkurve (bzw. das Drachenmuster
    in der Terminologie von Davis/Knuth). Aufruf (z. B.):
    >>> Drachenmuster(Papierfaltungszahl(5))
    Lösung für Aufgabe 8
    '''
    TurtleInit(s)
    laenge = int(200/(n*1.5))     # ggf. ändern
    turtle.forward(laenge)
    for i in range(len(s)):
        turtle.forward(laenge)
        if s[i] == '1':   turtle.left(90)  # Linksknick
        elif s[i] == '0': turtle.right(90) # Rechtsknick
        turtle.forward(laenge)
    turtle.forward(laenge)

def Drachenkurve(s):
    '''
    Eingabe: Papierfaltungszahl als String mit '0' und '1'
    Ausgabe: die entsprechende Drachenkurve (mit abgerundeten Ecken)
    Um z. B. die Verzierung an der Mauer des Knuth'schen Hauses
    nachzuvollziehen, ruft man
    >>> Drachenkurve(Papierfaltungszahl(9))
    auf. Lösung für Aufgabe 9
    '''
    TurtleInit(s)
    radius = int(200/(n*1.5))     # ggf. ändern
    for i in range(len(s)):
        if s[i] == '1':   turtle.circle(radius, 90)  # Linkskurve
        elif s[i] == '0': turtle.circle(-radius, 90) # Rechtskurve
```

Aufgabe 5: Klären Sie folgende Frage: Ist die Papierfaltungszahl rational oder irrational (Begründung!)?

In der Dissertation von Reimund Albers (http://www.mevis-research.de/~albers/Publikationen/Dissertation/index.html) wird bewiesen, dass die Papierfaltungsfolge aus L und R nicht periodisch ist (Kapitel 5, Satz 5.1, S. 73). Damit ist auch bewiesen, dass die Papierfaltungszahl irrational sein muss.

Die Papierfaltungszahl kann auch durch endliche Automaten erzeugt werden

Aufgabe 6: Entwerfen Sie einen endlichen Automaten, der die Nachkommastellen der Papierfaltungszahl erzeugt.

Zwei mögliche Lösungen findet man z. B. bei in der Dissertation von Reimund Albers, Kapitel 7 (http://www.mevis-research.de/~albers/Publikationen/Dissertation/index.html), vgl. auch J. Giesen, Aufgabe 43 in http://www.jgiesen.de/Divers/PapierFalten/Aufgaben/AufgabenTeil2.html.

Die im Artikel wiedergegebene Konstruktionsvorschrift lässt sich sehr elegant in eine rekursive Python-Funktion umsetzen, die die Turtle-Grafik benutzt:

```
from turtle import *

def dragon(step, length, zig=right, zag=left):
    if step <= 0:
        forward(length)
        return

    step -= 1
    length /= 1.41421

    zig(45)
    dragon(step, length, right, left)
    zag(90)
    dragon(step, length, left, right)
    zig(45)
```

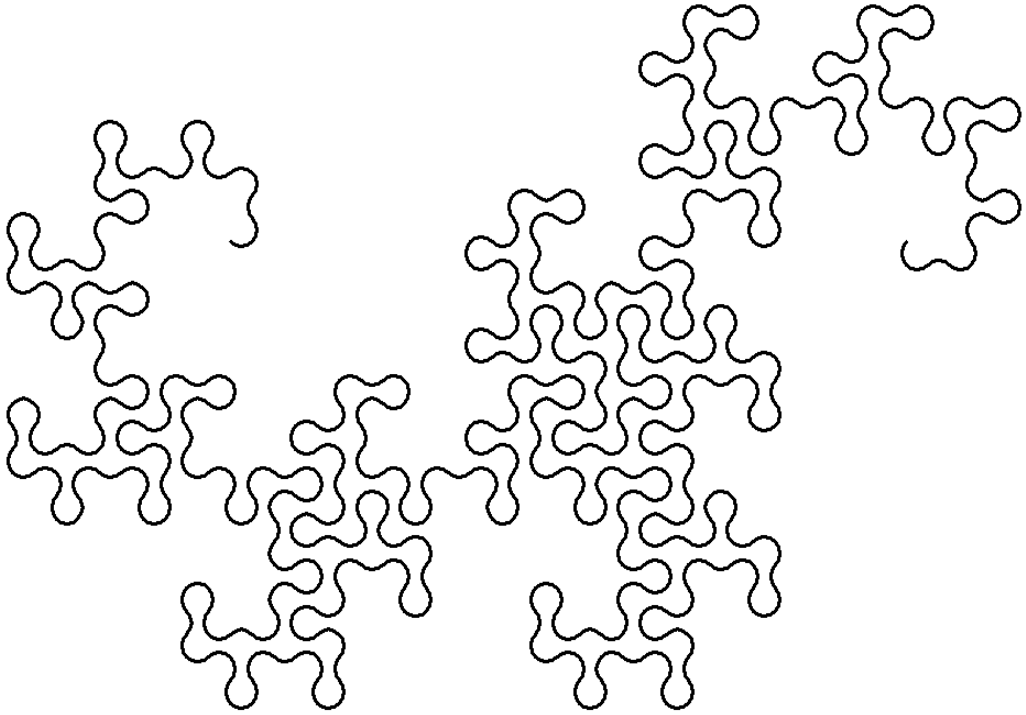
Aufgabe 7: Testen Sie das abgedruckte Python Programm. Sie erhalten den Python-Interpreter kostenfrei für alle gängigen Betriebssysteme unter www.python.org. Nach dem Starten des Programms wird nach dem Aufruf von z. B. `dragon(9,100)` die neunte Iterationsstufe der Drachenkurve gezeichnet, wobei die nullte Iterationstufe (die gerade Linie) eine Länge von 100 Pixeln hat.

Aufgabe 8: Implementieren Sie ein Programm zum Zeichnen der Drachenkurve in einer Programmiersprache Ihrer Wahl.

Das oben angegebene rekursive Python-Programm stammt von der Seite http://www.rosettacode.org/wiki/Dragon_curve, dort finden sich auch Implementierungen in vielen anderen Programmiersprachen. Ein Java-Programm und ein Applet zur Drachenkurve von Horst Gierhardt kann auf der Seite http://www.oberstufeninformatik.de/info11/turtle/Drachenkurve.html heruntergeladen werden. Eine schnelle iterative Lösung ist weiter oben angegeben (nach Aufgabe 4)

Aufgabe 9: Modifizieren Sie Ihr Programm so, dass die Drachenkurve mit abgerundeten Ecken ausgegeben wird.

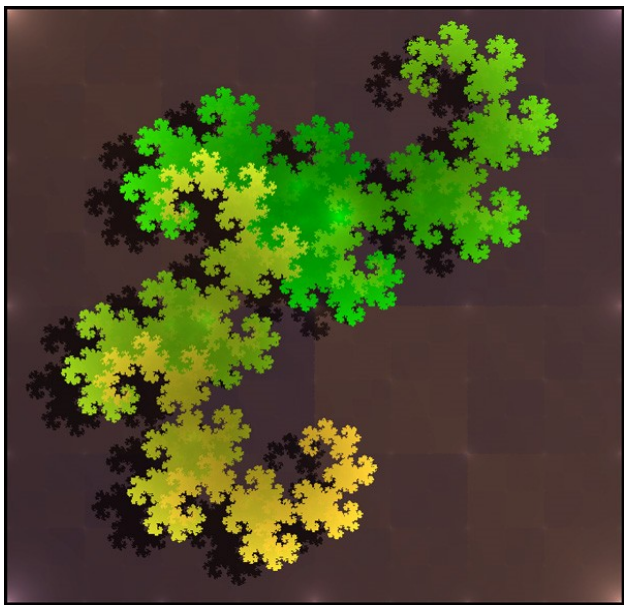
Siehe Python-Programm weiter oben (nach Aufgabe 4). Damit kann auch die Dekoration am Haus des Ehepaars Knuth nachempfunden werden (s. die folgende Abbildung).



Drachenkurve mit abgerundeten Ecken zur 9. Iterationsstufe (Bild erzeugt mit dem o. a. Programm)

Aufgabe 10: Begründen Sie folgende Aussage: Echte Selbstähnlichkeit liegt nur beim Drachenfraktal vor, bei der Drachenkurve gilt die Selbstähnlichkeit nur näherungsweise (s. Abb. 12).

Die näherungsweise Selbstähnlichkeit der Drachenkurve ergibt sich unmittelbar aus Bild 10: die beiden kongruenten Drachenkurven der 11. Stufe sehen der Drachenkurve der 12. Stufe schon ziemlich ähnlich, sind aber zu dieser nicht kongruent. Entsprechend kann man sich die Drachenkurve der 11. Stufe jeweils aus zwei Drachenkurven der 10. Stufe zusammengesetzt denken usw. Je höher die Iterationsstufe wird, desto mehr ähneln sich die Drachenkurven bei aufeinander folgenden Stufen. Im Grenzübergang, wenn die Zahl der Iterationsstufen gegen unendlich geht, werden die Teilfiguren tatsächlich kongruent zur Gesamtfigur. Damit wird erst bei der Grenzfigur, dem Drachenfraktal, echte Selbstähnlichkeit erreicht (s. die folgende Abb. aus http://en.wikipedia.org/wiki/Dragon_curve).



H. Witten, 13.12.2008