# Aufgabenblatt Rekursion

## Einteilung der Aufgaben:

- Einstieg: Von allen zu lösende Aufgaben, um die Kernideen zu verstehen (Pflicht).
- Übung: weitere Aufgaben zum Üben ohne wesentlich neue Elemente.
- Ausblick: weiterführende Aufgaben mit zusätzlichen Fragestellungen.

## Aufgabe 1 (Einstieg)

Die folgenden beiden Programme nutzen rekursive Funktionen. Gehen Sie die Programme zuerst einmal durch und stellen Sie sicher, dass Sie verstehen, wie diese arbeiten.

- Wo ist da jeweils der Basisfall?
- Wie wird in a) sichergestellt, dass der Basisfall auch erreicht wird?
- Was geschieht in beiden Fällen, wenn die Funktion mit nicht-ganzzahligen Werten aufgerufen wird?
- Notieren Sie für die Aufgabe b) die Ausgabe für collatz(3).

a)

```
1 def addiere(a, b):
2     if b == 0:
3         return a
4     else:
5         return addiere(a+1, b-1)
6
7 print(addiere(5, 7))
```

b)

```
1 def collatz(n):
2     print(n)
3     if n>1:
4         if n % 2 == 0:
5             a = n//2
6         else:
7             a = 3*n+1
8         collatz(a)
```

## Aufgabe 2 (Einstieg)

Ein häufiger Fall, indem auf rekursive Programmiertechniken zurückgegriffen wird, ist das Durchsuchen von sogenannt baumähnlichen Strukturen, wie beispielsweise das Dateiverzeichnis eines Computers.

Wie würden Sie vorgehen, wenn Sie in Ihrem Dateiverzeichnis eine Datei suchen würden, von der Sie nicht mehr wissen, in welchen Unterordner Sie diese abgelegt haben?

Dabei stehen Ihnen nur folgende Werkzeuge zur Verfügung:

- Durchsuchen eines Ordners nach einem bestimmten Dateinamen.
- Auflisten aller Unterordner eines Ordners.

Definieren Sie auf dieser Grundlage eine Funktion suche(datei, ordner). Es reicht, wenn Sie diese Funktion in deutscher Sprache in Form von Anweisungen formulieren und durch Einzüge so gestalten, dass man das Vorgehen rasch versteht.

## Aufgabe 3 (Einstieg)

Nehmen wir an, Sie hätten eine Liste, die als Elemente sowohl Zahlen oder Strings als auch Listen enthält, die wiederum die gleichen Elemente enthalten können. Beispielsweise folgende Liste:

```
a = [1, [2, 3], ["vier", [5, 6], [7, ["acht", 9]], 10], "elf", [12, 13]]
```

Schreiben Sie eine Funktion, die alle Elemente der Liste selbst und aller ihrer Unterlisten und Unterunterlisten etc. nacheinander in der Shell ausgibt. Die Listen selbst sollen dabei nicht ausgegeben werden, sondern nur jene Elemente, die selbst nicht Listen sind.

Tipp: Um zu prüfen, ob ein Objekt eine Instanz von einem bestimmten Typ ist, nutzen Sie die Funktion `isinstance(objekt, typ)`.

## Aufgabe 4 (Einstieg/Übung)

Sie haben in den Folien gesehen, wie man die Multiplikation durch eine rekursive Vorgehensweise nur mit der Addition zweier Zahlen ausführen kann. In Der Aufgabe 1a) haben Sie zudem gesehen, dass die Addition nur mit +1 und -1 berechnet werden kann, wenn man Rekursion nutzt. Auch andere Rechenarten können rekursiv definiert werden, indem dabei nur einfachere Rechnungsarten genutzt werden. Dazu folgende zwei Aufgaben:

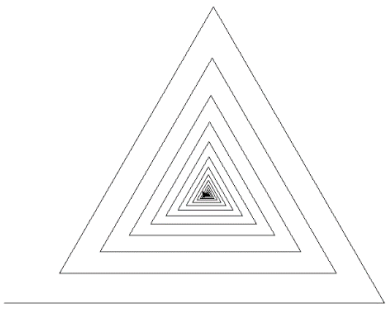
a) **(Einstieg)** Definieren Sie eine rekursiv arbeitende Funktion für das Potenzieren. Dabei soll `potenziere(a, b)` den Wert von $a^b$ zurück geben. Als Grundoperationen sollen nur die Multiplikation und die Subtraktion genutzt werden

b) **(Übung)** Definieren Sie eine rekursiv arbeitende Funktion für die Modulo-Operation. Dabei soll `modulo(a, b)` den Wert von $a\%b$ zurück geben.
Tipp: Sie brauchen die Subtraktion.

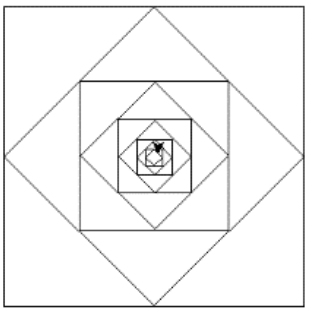
## Aufgabe 5 (Ausblick)

Mit Rekursion lassen sich auch einige grafische Aufgaben relativ leicht lösen. Hierzu drei Beispiele.

a) Folgende Grafik wurde mittels Rekursion erstellt. Dabei wurde jeweils nur eine Linie gezeichnet und ein Winkel gemacht. Anschliessend wurde die Funktion mit einer 10% kürzeren Linienlänge aufgerufen.
Versuchen Sie dies selbst zu programmieren und nutzen Sie dabei auch eine rekursive Vorgehensweise.
Vergessen Sie den Basisfall nicht.



b) Auch diese Grafik entstand mittels einer rekursiv arbeitenden Funktion. Versuchen Sie diese nachzuprogrammieren.



Tipp 1: In einem Quadrat gilt für die Diagonale d und die Seitenlänge s: $d = \sqrt{2} \cdot s$.
Tipp 2: Wurzeln lassen sich mittels Potenzen schreiben. Es gilt: $\sqrt{n} = n^{0.5}$.
Tipp 3: Nutzen Sie eine zweite Funktion, die Ihnen jeweils ein Quadrat mit einer bestimmten Seitenlänge automatisch zeichnet.

c) **Herausforderung:**
Sie sehen unten ein sogenanntes Sierpiński -Dreieck.
Dieses geht zurück auf Wacław Sierpiński, der dieses Fraktal 1915 definiert hat.
In einem Dreieck wird dazu das Mitteldreieck eingezeichnet, das durch die Verbindung der Mittelpunkte der drei Seiten entsteht. Anschliessend wird in den so entstandenen äusseren drei Dreiecken der Vorgang jeweils wiederholt.
Viel Spass beim selbst programmieren!
Achtung: Denken Sie daran, eine Abbruchbedingung zu definieren.