Schachtelfiguren im Schnee: Rekursion mit Python


Manche Gegenstände oder Bilder kannst du am besten beschreiben, indem du dich bei der Beschreibung ständig wiederholst. Schau dir die Puppen auf den beiden Bildern an, sie kommen aus Russland und heißen auch Matrjoschka-Puppen (auf Deutsch »Mütterchen«). Solltest du jemandem, der die Bilder nicht sieht, erklären, was eine Matrjoschka-Puppe ist, dann könntest du zum Beispiel sagen: »Eine Matrjoschka-Puppe ist eine Puppe, die eine Puppe enthält, die eine Puppe enthält«. So weit ok. Aber es gibt auch Matrjoschka-Puppen, da sind mehr als 10 Puppen ineinander verschachtelt. Dann müsstest du mehr als 10 Mal in einem Satz sagen »Das ist eine Puppe, die eine Puppe enthält, die eine Puppe enthält, …, die eine Puppe enthält«. Da sagst du dann vielleicht lieber: »Eine *Matrjoschka-Puppe* ist eine Puppe, die eine etwas kleinere *Matrjoschka-Puppe* enthält«. So erklärst du den Begriff Matrjoschka-Puppe, indem du in der Erklärung den Begriff selbst wieder verwendest. Clever!

Aber geht das überhaupt? Wenn du genau drüber nachdenkst, dann ist die Erklärung für alle Puppen richtig, außer für die allerkleinste ganz innen. Denn die ist, korrekt gesehen, keine Matrjoschka-Puppe, sondern einfach nur ein Holzpüppchen, was man nicht aufmachen kann.

Schachtelfiguren

Ein neues Beispiel: Unten siehst du ein Quadrat aus lauter Quadraten, die ineinander geschachtelt sind, nennen wir es ein *Schachtelquadrat*. Die Beschreibung ähnelt sehr stark der Beschreibung der Matrjoschka-Puppen: »Ein Schachtelquadrat ist ein Quadrat, das ein etwas kleineres Schachtelquadrat enthält«. Auch hier gilt: Aufgepasst beim kleinsten Quadrat, denn dort ist gar nichts enthalten.

Im Gegensatz zu den Matrjoschka-Puppen kannst du ein Schachtelquadrat sehr einfach mit der Schildkröte zeichnen. In der Funktion dafür musst du folgendes machen:

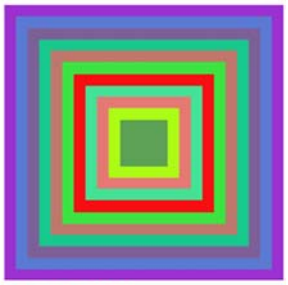
1. Zeichne ein Quadrat (mit einem Parameter für die Seitenlänge), fange dazu bei der linken oberen Ecke an und ende dort auch.
2. Gehe auf die linke obere Ecke des nächstkleineren Quadrats (zum Beispiel 10 Schritte nach rechts und 10 nach unten).
3. Rufe wieder dieselbe Funktion bei Schritt 1 auf, aber mit einem kleineren Wert für die Seitenlänge, um das nächstkleinere Quadrat zu zeichnen.

4. ABER AUFGEPASST: Wenn das kleinste Quadrat erreicht wird, dann hörst du auf, Schritt 3 auszuführen.

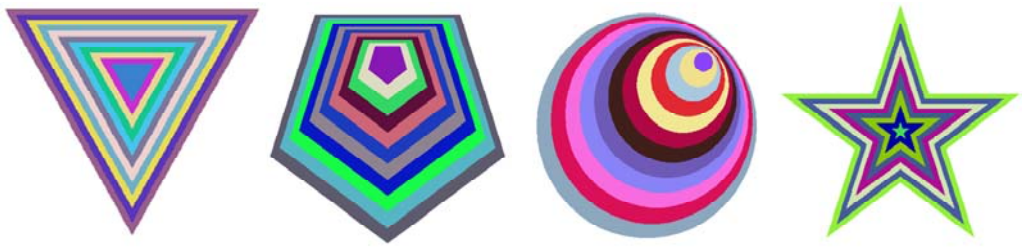
Weil wir beim Programmieren alles immer ganz korrekt beschreiben müssen (denn Computer denken ja nicht mit), ist es enorm wichtig, das allerkleinste Quadrat extra zu behandeln, denn sonst endet unser Programm nie, weil die Funktion sich bis in alle Ewigkeit selbst aufruft. Für die Schritte 3 und 4 brauchen wir daher eine bedingte Anweisung.

Unten siehst du die Funktion für das Schachtelquadrat (Datei *Schachtelquadrat.py*). Die dort benutzte Funktion `Quadrat(Seitenlänge)` zeichnet einfach ein Quadrat mit der angegebenen Seitenlänge, wobei vorher die Farbe auf eine Zufallsfarbe gestellt wurde. In dieser Farbe wird das Quadrat dann auch ausgemalt. Das schaffst du bestimmt alleine. Die Funktion Schachtelquadrat rufst du am besten mit einer Seitenlänge von mehr als 200 Pixeln auf. Das nächstinnere Schachtelquadrat ist dann immer 20 Pixel kleiner als sein äußerer Nachbar. Das kleinste Quadrat soll eine Seitenlänge von höchstens 20 Pixeln haben.

```
def Schachtelquadrat(Seitenlänge):
  # 1. Male das aktuelle Quadrat
  Quadrat(Seitenlänge)
  # 2. Nimm Position für das nächstinnere Quadrat ein
  penup()
  forward(10)
  right(90)
  forward(10)
  left(90)
  pendown()
  # Enthält das aktuelle Quadrat noch weitere Schachtelquadrate?
  if Seitenlänge-20 > 20:
  # 3. Falls ja: Weitermachen mit nächstkleinerem Schachtelquadrat
    Schachtelquadrat(Seitenlänge - 20)
  # 4. Falls nicht: Aufhören mit einem einfachen Quadrat
  else:
    Quadrat(Seitenlänge - 20)
```



Bekommst du das hin? Passe die Schachtelquadrat-Funktion so an, dass du auch Schachteldreiecke, Schachtelfünfecke, Schachtelkreise und Schachtelfünfsterne erzeugen kannst. Bei unseren Versuchen im Bild sind die Verschachtelungen manchmal nicht ganz mittig, was auch lustig aussieht. Die Lösungen sind in der Datei *Schachtelfiguren.py* enthalten.



Profiprogrammierer bezeichnen Funktionen, die sich selber aufrufen, als *Rekursion*. Auf Deutsch heißt das Wiederauftauchen oder Wiederkommen,

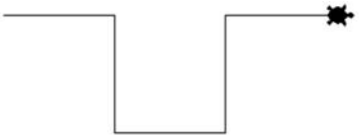
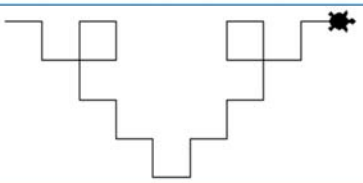
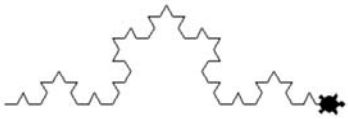
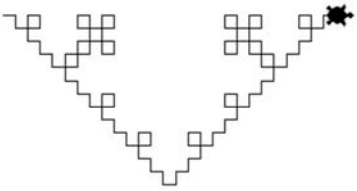
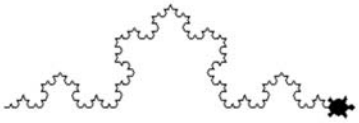
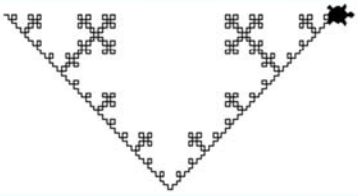
Es schneit

Vom Frühling mit blühenden Bäumen begeben wir uns jetzt in den Winter. Dir ist bestimmt schon einmal aufgefallen, wie schön eine Schneeflocke von nahem aussieht. Schneeflocken sind Eiskristalle. Alle Kristalle haben eine regelmäßig aufgebaute Form. Die Grundform ist ein Vieleck, zum Beispiel ein Dreieck oder Fünfeck. Hättest du eine Lupe und eine nicht-schmelzende Schneeflocke, so könntest du sehen, dass die Seitenränder nicht glatt sind, sondern Zacken nach außen oder nach innen haben. Wenn du dann in so eine Zacke hineinzoomst, siehst du, dass auch die nicht aus glatten Linien besteht, sondern aus gezackten. Ahnst du es schon? Hier schreit es schon wieder nach einer rekursiven Funktion.

Wir schauen uns jetzt einen Seitenrand einer Schneeflocke genauer an. Von weitem betrachtet (RT=0) ist so ein Rand eine gerade Linie. Nun zoomst du einmal auf die Linie (RT=1) und erkennst: Die Linie hat eine Zacke. Sie besteht eigentlich aus vier Linienstücken, die eine Spitze nach oben bilden. Zoome auf ein beliebiges dieser Linienstücke, aus denen die Zacke besteht (RT=2) und du erkennst: Jedes dieser Linienstücken hat wieder eine Zacke und besteht eigentlich aus vier Linienstücken, die eine Spitze nach oben bilden. Wie es weitergeht, kannst du dir denken.

Die Tabelle zeigt links deine Zoom-Aktionen für eine Linie mit spitzer Zacke nach oben. In jedem Level wird jede der bisherigen Linien wieder gezackt. In Level RT=4 siehst du, dass die vielen neuen Zacken dazu führen, dass das Gebilde eher runder und flockiger als spitzer wird.

Die rechte Spalte der Tabelle zeigt die Effekte, die entstehen, wenn die Linie nicht durch eine Zacke nach oben ersetzt wird, sondern durch ein stumpfes Rechteck nach unten. Obwohl eigentlich nur Kleinigkeiten anders sind als bei der spitzen Zacke, sehen die Bilder komplett unterschiedlich aus – und unerwartet schön.

RT	Spitze Zacke nach oben	Stumpfes Rechteck nach unten
0		
1		
2		
3		
4		

Programmiere nun die Bilder in der linken Tabellenspalte: In der rekursiven Funktion `SeiteSpitzNachOben(Schritte, RT)` machst du folgendes:

1. Rufe dreimal (für jedes Linienstück) die Funktion `SeiteSpitzNachOben` auf, wobei der Wert für den Parameter Schritte ein Drittel der Seitenlänge der zu unterteilenden Linie ist. Vor jedem der drei Aufrufe drehst du die Schildkröte in die richtige Richtung. Bei jedem Aufruf verringerst du die Rekursionstiefe RT um eins.
2. Ist die Rekursionstiefe RT gleich 0, dann endet die Rekursion. Wenn diese Zahl erreicht ist, malst du nur noch eine gerade Linie, anstatt die Funktion weiter aufzurufen.

Hier ist der Programmcode für die Funktion `SeiteSpitzNachOben(Schritte, RT)`. Die Funktion findest du in der Datei *Schneeflocke.py*.

```
def SeiteSpitzNachOben(Schritte, RT):
    if RT > 0:
        # mache aus der Seite eine Spitze (mit 4 Seiten)
        SeiteSpitzNachOben(Schritte/3, RT-1)
        left(60)

        SeiteSpitzNachOben(Schritte/3, RT-1)
        right(120)

        SeiteSpitzNachOben(Schritte/3, RT-1)
        left(60)

        SeiteSpitzNachOben(Schritte/3, RT-1)

    else:
        # Male eine gerade Linie der Länge Schritte
        forward(Schritte)
```

Bekommst du das hin? Wandle die Funktion `SeiteSpitzNachOben` ab zu Funktionen `SeiteSpitzNachUnten`, `SeiteStumpfNachOben` und `SeiteStumpfNachUnten`. Sieh dir Kurven an, die mit den neuen Funktionen erstellt wurden. Welche gefallen dir am besten?

Koch-Kurve

Die Konstruktion in der linken Spalte unserer Tabelle heißt »Koch-Kurve«, denn sie wurde 1904 vom schwedischen Mathematiker Helge von Koch entdeckt. Damals wurde sie auch »Monster-Kurve« genannt.

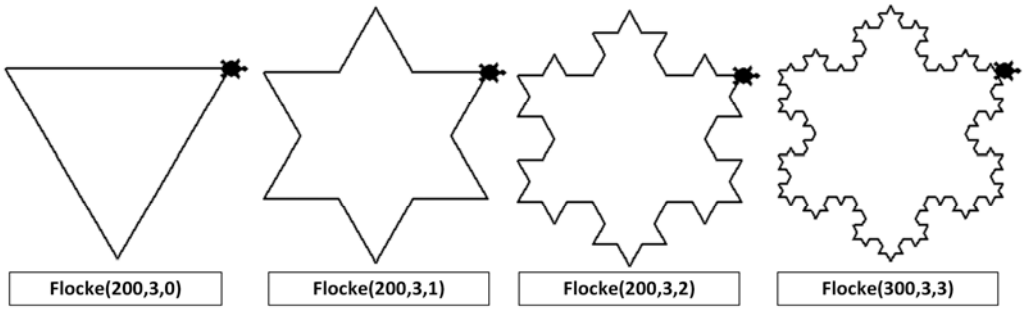
Bisher haben wir uns nur Ränder von Schneeflocken angesehen, die sich beim Zoomen von Linien zu wunderschönen Kurven verwandeln. Nun wollen wir diese Seitenränder zu Schneeflocken zusammensetzen, damit es endlich schneien kann. Die Zusammensetzung ist gar nicht schwer. Du benutzt wieder die Vieleckfunktion (die du ja schon kennst). Es gibt nur einen Unterschied zu früher, als du Vieleck-Seiten durch gerade Strecken beschrieben hast (mit der forward-Anweisung): du rufst stattdessen jetzt für jede Strecke die rekursive Funktion auf, die die Zerlegung der Strecke durch spitze Zacken oder stumpfe Rechtecke beschreibt.

Hier ist die Flocken-Funktion, die die rekursive Funktion `SeiteSpitzNachOben` benutzt (Datei *Schneeflocke.py*):

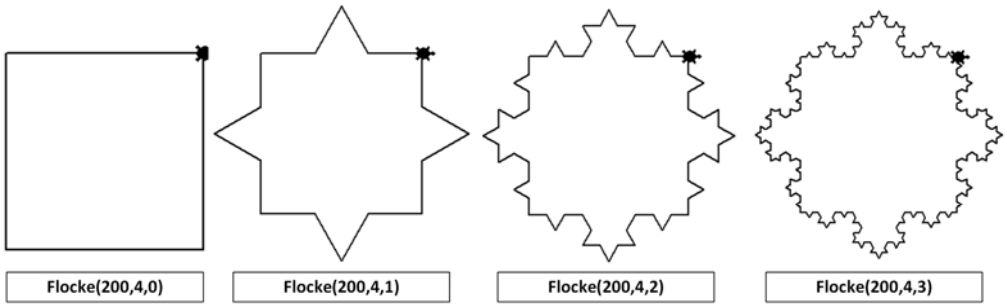
```
def Flocke(Schritte, Ecken, RT):
   for i in range(Ecken):
```

```
right(360/Ecken)
SeiteSpitzNachOben(Schritte, RT)
```

Und hier sind ein paar Ergebnisse: In der ersten Bilderreihe siehst du Flocken mit dreieckiger Grundform in verschiedenen Rekursionstiefen (die nennt man auch »Koch'sche Schneeflocken«, weil sie die Koch-Kurve verwenden).



Den Flocken in der zweiten Bilderreihe liegt ein Quadrat zugrunde.



Bekommst du das hin? Sieh dir weitere Schneeflocken an, die du mit Vielecken und den Funktionen `SeiteSpitzNachOben`, `SeiteSpitzNachUnten`, `SeiteStumpfNachOben` und `SeiteStumpfNachUnten` erstellst. Welche gefallen dir am besten?

Das Bild zeigt dir eine davon, unsere Lieblingsschneeflocke. Wie wurde sie erzeugt?