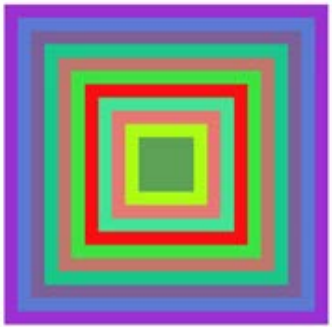
7. Schachtelquadrat

Nun haben wir alles zusammen, um rekursive Muster malen zu können. Beginnen wir mit einem Quadrat, was innen weitere Quadrate enthält – genannt Schachtelquadrat:



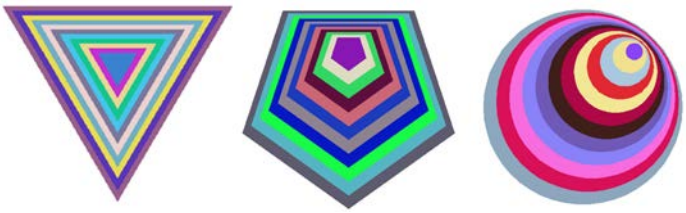
Im Gegensatz zu Matrjoschka-Puppen kannst du ein Schachtelquadrat sehr einfach mit der Schildkröte zeichnen. Du brauchst eine Funktion *Schachtelquadrat* mit einem Parameter *Seitenlänge* In der Funktion *Schachtelquadrat(Seitenlänge)* musst du folgendes machen:

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 die Funktion *Schachtelquadrat* auf, aber mit einem kleineren Wert für die Seitenlänge, um das nächstkleinere Quadrat zu zeichnen.
4. **ABER** [Abbruchbedingung]: Wenn das kleinste Quadrat erreicht wird, dann hörst du auf, Schritt 3 auszuführen.

Als Abbruchbedingung ist es enorm wichtig, das allerkleinste Quadrat extra zu behandeln, denn sonst endet unser Programm nie, weil die Funktion *Schachtelquadrat*() sich bis in alle Ewigkeit selbst aufruft. Für die Schritte 3 und 4 brauchst du daher eine *if-else*-Bedingung.

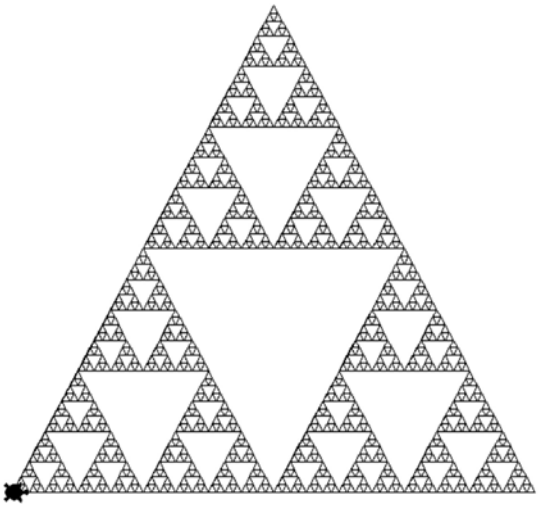
```
def Schachtelquadrat(Seitenlänge):
    # 1. Male das aktuelle Quadrat
    Quadrat(Seitenlänge)
    # 2. Geh auf Position für das nächst-kleinere Quadrat
    …  # hier Code einfügen
    # Prüfe: Enthält das aktuelle Quadrat noch weitere Schachtelquadrate?
    if Seitenlänge > 40:
        # 3. Wenn ja, dann hier nächstkleineres Schachtelquadrat malen
        Schachtelquadrat(Seitenlänge – 20)
    else:
        # 4. Sonst: Aufhören mit einem einfachen Quadrat (dem kleinsten)
        Quadrat(Seitenlänge – 20)
```

Bekommst du das hin? Passe die Schachtelquadrat-Funktion so an, dass du auch Schachteldreiecke, Schachtelfünfecke und Schachtelkreise erzeugen kannst. Bei unseren Versuchen im Bild sind die Verschachtelungen manchmal nicht ganz mittig, was auch lustig aussieht.



8. Sierpinsky-Dreieck

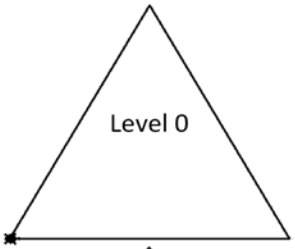
Eine sehr schöne zusammengesetzte Figur, die nur aus Dreiecken besteht, ist das sogenannte *Sierpinski-Dreieck*, benannt nach seinem Erfinder Wacław Sierpiński, der die Figur 1915 zum ersten Mal beschrieben hat.



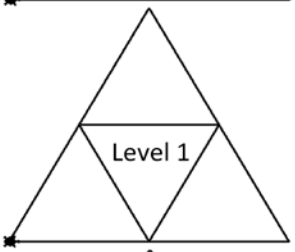
Die Frage ist, wie man diese Figur am besten beschreibt. Die Grundform scheint ein Dreieck zu sein, was wieder in weitere Dreiecke zerlegt wird.

Machen wir uns das klar anhand von Bildern von Sierpinski-Dreiecken mit unterschiedlich vielen Zerlegungen. Man nennt die Anzahl der Zerlegungen auch Rekursionstiefe (oder einfach »Level«).

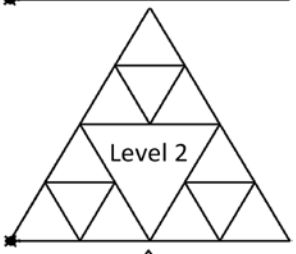
Level 0. Hier gibt es überhaupt keine Zerlegung des Dreiecks. Die Definition lautet einfach nur »Ein Sierpinski-Dreieck in Level 0 ist ein auf der Seite liegendes gleichseitiges Dreieck. Punkt«.



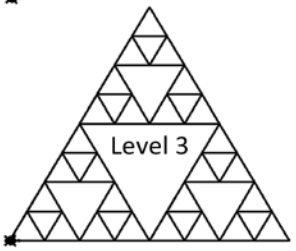
Level 1. In diesem Level wird das äußere Dreieck einmal in drei innere auf der Seite liegende Dreiecke zerlegt. Die Beschreibung lautet: »Ein Sierpinski-Dreieck in Level 1 ist ein auf der Seite liegendes gleichseitiges Dreieck, was in seinen drei Spitzen je ein Dreieck mit halber Seitenlänge enthält«.



Level 2. Die Definition enthält jetzt zwei Zerlegungsschritte (und wird damit noch länger). Nun werden die inneren Dreiecke auch noch zerlegt. Die Definition lautet: »Ein Sierpinski-Dreieck in Level 2 ist ein auf der Seite liegendes gleichseitiges Dreieck, was in seinen drei Spitzen je ein Dreieck mit halber Seitenlänge enthält, welches in seinen drei Spitzen je ein Dreieck mit halber Seitenlänge enthält«.



Level 3. Ein Level schaffen wir noch, diesmal mit drei Zerlegungsschritten. Wieder kommt die Zerlegung der bisher innersten Dreiecke dazu: »Ein Sierpinski-Dreieck in Level 3 ist ein auf der Seite liegendes gleichseitiges Dreieck, was in seinen drei Spitzen je ein Dreieck mit halber Seitenlänge enthält, welches in seinen drei Spitzen je ein Dreieck mit halber Seitenlänge enthält, welches in seinen Spitzen je ein Dreieck mit halber Seitenlänge enthält«.



Langsam wird es mühsam. Aber das Prinzip ist klar und jetzt bekommen wir die vollständige rekursive Definition hin, die für alle möglichen Rekursionstiefen (Zerlegungsschritte) gilt: »**Ein Sierpinski-Dreieck ist ein auf der Seite liegendes gleichseitiges Dreieck, was in seinen drei Spitzen je ein weiteres Sierpinski-Dreieck mit halber Seitenlänge enthält.** « So einfach ist das. Es fehlt nur noch das Rekursionsende: »**Das allerkleinste Sierpinski-Dreieck ist ein einfaches Dreieck**«.

Und genauso einfach wollen wir nun mit dir ein Sierpinski-Dreieck programmieren. Wie gehst du vor? Du definierst folgende Funktion `Sierpinski(Seitenlänge, RT)`:

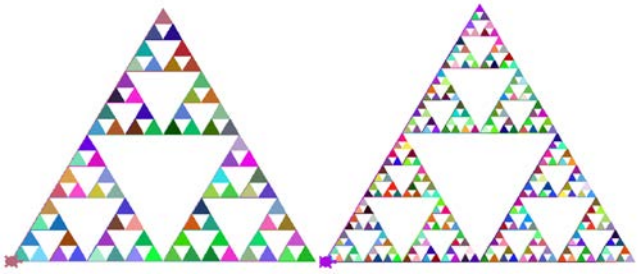
1. Du rufst dreimal (für jedes der in den Spitzen liegenden Dreiecke) die Funktion *Sierpinski* auf, wobei der Wert für den Parameter Seitenlänge die Hälfte der Seitenlänge des äußeren Dreiecks ist. Vor jedem der drei Aufrufe positionierst du die Schildkröte in der linken unteren Ecke des neu zu zeichnenden Sierpinski-Dreiecks.
2. Die Rekursion muss irgendwann enden. Hierfür verwendest du den Parameter RT, der die Rekursionstiefe zählt. Er wird bei jedem Funktionsaufruf um eins verringert. Du hörst auf, wenn RT=0 erreicht ist. Dann malst du ein einzelnes Dreieck, anstatt die Funktion weiter aufzurufen.

Hier ist der Programmcode der Sierpinski-Funktion (Datei *Sierpinski.py*). Du rufst die Funktion auf zum Beispiel mit `Sierpinski(400,4)`. Probiere mal verschiedene Rekursionstiefen aus.

```
def Sierpinski(Seitenlänge,RT):
    if RT > 0:
    # 1. Male drei Sierpinski-Dreiecke in die drei Ecken
        Sierpinski(Seitenlänge/2,RT-1) # erstes Dreieck links unten

        forward(Seitenlänge/2)         # gehe zu Mitte der unteren Seite

        Sierpinski(Seitenlänge/2,RT-1) # zweites Dreieck rechts unten

        backward(Seitenlänge/2)        # gehe zur Mitte der linken Seite
        left(60)
        forward(Seitenlänge/2)
        right(60)

        Sierpinski(Seitenlänge/2,RT-1) # drittes Dreieck oben

        left(60)                       # zurück auf Anfang links unten
        backward(Seitenlänge/2)
        right(60)
    else:
    # 2. Die Rekursion endet: Male nur noch ein einfaches Dreieck
        for i in range(0,3):
            forward(Seitenlänge)
            left(120)
```

Bekommst du das hin?

Nutze deine Kenntnisse über Zufallsfarben und erzeuge bunte Sierpinski-Dreiecke, wie hier im Bild zu sehen.



9. Bäume und Wälder

Auch in der Natur kommen Gebilde vor, die aus immer kleineren Varianten von sich selbst bestehen. Ein typisches Beispiel ist ein Baum. Er beginnt unten mit einem Stamm, der sich nach oben zu einer Astgabel verzweigt, deren Enden sich nach oben zu Astgabeln verzweigen, deren Enden sich nach oben zu Astgabeln … du ahnst, wie es weitergeht.

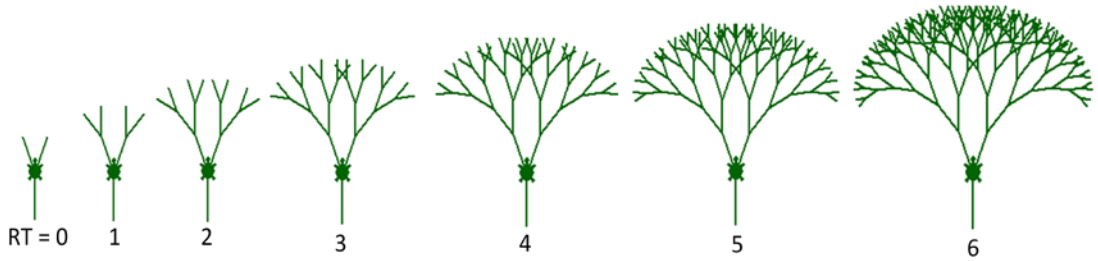
Hörst du es auch? Bäume schreien geradezu nach einer Beschreibung als rekursive Funktion. Mittlerweile hast du ja darin schon etwas Übung. Du gibst wieder eine Rekursionstiefe RT vor, die die Tiefen der Astgabelverzweigungen zählt. Nehmen wir an, jeder Ast gabelt sich nach oben in 2 Unteräste (wie ein Ypsilon »Y«). Du definierst folgende Funktion `Baum(Astlänge, Winkel, RT):`

1. Du malst einen Stamm (eine Linie) von unten nach oben.
2. Du rufst zweimal (für jede Spitze der Astgabelung im aktuellen Level) die Funktion Baum auf, wobei der Wert für den Parameter *Astlänge* 8/10 der Astlänge des aktuellen Baumes ist, damit die Äste nach oben etwas kürzer werden. Du drehst vor dem ersten Aufruf die Schildkröte ein Stück nach links (Parameter *Winkel*), um die linke Spitze der Astgabel zu malen. Vor dem zweiten Aufruf drehst du die Schildkröte um das doppelte dieses Winkels zurück nach rechts, um die rechte Spitze der Astgabel zu bekommen. Vergiss nicht, bei jedem Aufruf der Funktion Baum die Rekursionstiefe (Parameter *RT*) um eins zu verringern.
3. Ist die Rekursionstiefe RT gleich 0, dann endet die Rekursion. Wenn diese Zahl erreicht ist, malst du nur noch eine einzige Astgabel, anstatt die Funktion weiter aufzurufen.

Die Funktion `Baum(Winkel, Astlänge, RT)` rufst du zum Beispiel mit `Baum(20, 50, 4)` auf. Probiere mal verschiedene Winkel, Astlängen und Rekursionstiefen aus.

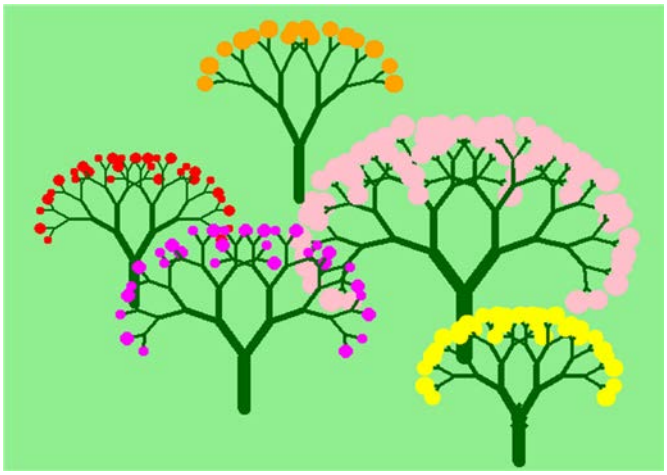
```
def Baum(Winkel, Astlänge, RT):
    # 1. Male einen Ast (den Stamm der Gabelung)
    forward(Astlänge)
    # 2. Male die Astgabelung: Rufe zweimal Baum auf
    if RT > 0:
        left(Winkel)
        Baum(Winkel, 0.8*Astlänge, RT-1) # linker Teil der Gabelung

        backward(0.8*Astlänge)

        right(Winkel*2)
        Baum(Winkel, 0.8*Astlänge, RT-1) # rechter Teil der Gabelung

        backward(0.8*Astlänge)
        left(Winkel)
    else:
   # 3. male die letzte Astgabelung an der Spitze
        left(Winkel)
        forward(0.8*Astlänge)
        backward(0.8*Astlänge)
        right(Winkel*2)
        forward(0.8*Astlänge)
        backward(0.8*Astlänge)
        left(Winkel)
```

Im Bild siehst du Bäume, die bei Aufrufen mit unterschiedlichen Rekursionstiefen entstehen, aber alle mit gleichen Winkeln und gleicher Astlänge. Es sieht aus wie in einer Baumschule mit jungen und alten Bäumen.



Die Bäume, die du mit der Baum-Funktion malen kannst, sind hier allerdings noch ein bisschen kahl. Es ist eine Baumschule im Winter.

Bekommst du das hin? Male an die Spitzen der letzten Astgabeln grüne Blätter. Dafür kannst du die Anweisung `dot(Dicke,Farbe)` benutzen, da hinterlässt die Schildkröte einen Klecks, dessen Dicke du als ersten Parameter in der Klammer angibst (kleinste Dicke = 1). Der zweite Parameter ist die Farbe, zum Beispiel `"darkgreen"`. Natürlich kannst du die Kreise auch andersfarbig ausmalen, dann sieht es so aus wie auf dem Bild mit blühenden Bäumen im Frühling. Male mehrere solcher Bäume an verschiedenen Positionen.



10. Schneeflocken

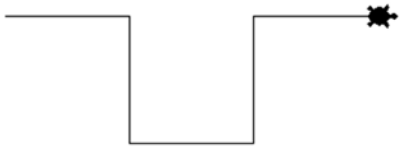
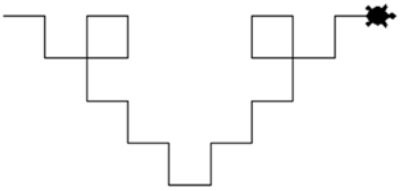
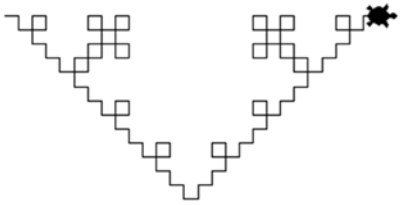
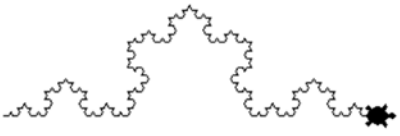
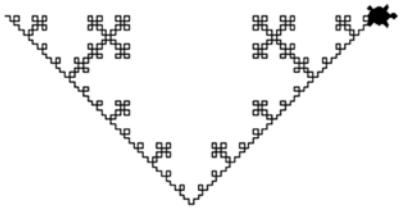
Vom Frühling mit blühenden Bäumen begeben wir uns jetzt in den Winter. Dir ist bestimmt schon einmal aufgefallen, wie schön eine Schneeflocke von nahem aussieht. Schneeflocken sind Eiskristalle. Alle Kristalle haben eine regelmäßig aufgebaute Form. Die Grundform ist ein Vieleck, zum Beispiel ein Dreieck 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 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 Teilstück) 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):`

```
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

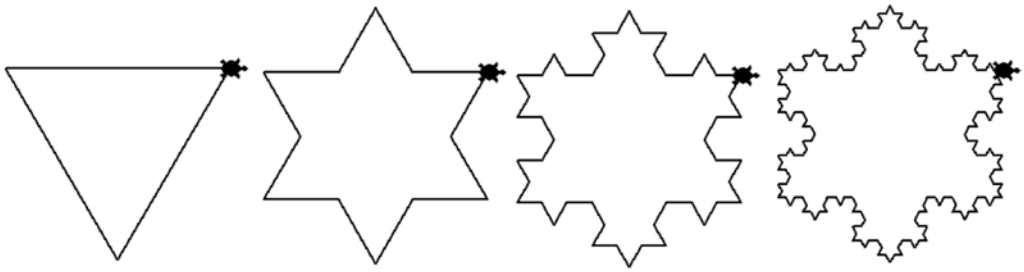
Die Konstruktion in der linken Spalte unserer Tabelle heißt »Koch-Kurve«, denn sie wurde 1904 vom schwedischen Mathematiker Helge von Koch entdeckt. Damals wurde sie auch »Monster-Kurve« genannt.

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 beispielsweise eine Dreiecksfunktion. Es gibt nur einen Unterschied zu früher, als du Dreieck-Seiten durch gerade Strecken beschrieben hast (mit der *forward*-Anweisung und der Variablen *Schritte* für die Länge der Strecke): du rufst stattdessen jetzt für jede Strecke die rekursive Funktion auf, die die Zerlegung der Strecke durch spitze Zacken oder stumpfe Rechtecke beschreibt.

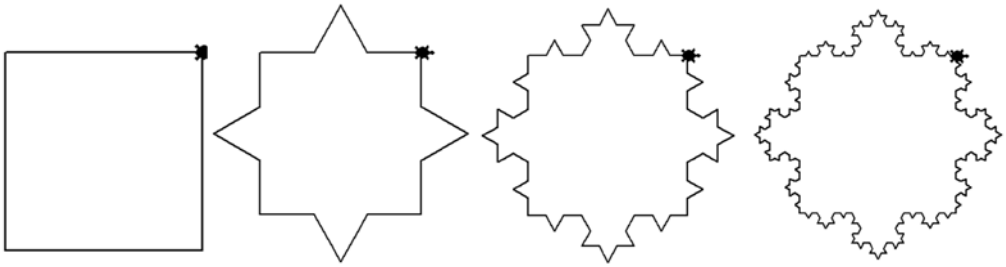
Hier ist die Flocken-Funktion, die die rekursive Funktion `SeiteSpitzNachOben` benutzt:

```
def Flocke(Schritte, RT):
    # Grundform ist ein Dreieck, also 3 Linien
    for i in range(3):
    # Statt Linien male Schneeflockenseiten
    SeiteSpitzNachOben(Schritte, RT)
    right(120)
```

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).

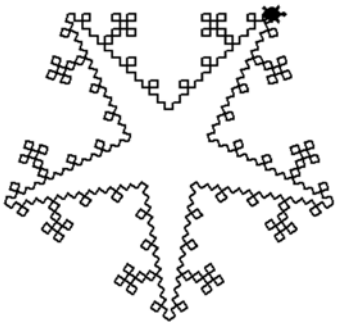


Den Flocken in der zweiten Bilderreihe liegt ein Quadrat zugrunde.



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



Rätsel: Finde die Schildkröte im Schnee ☺: