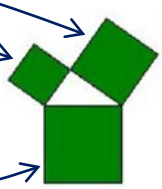
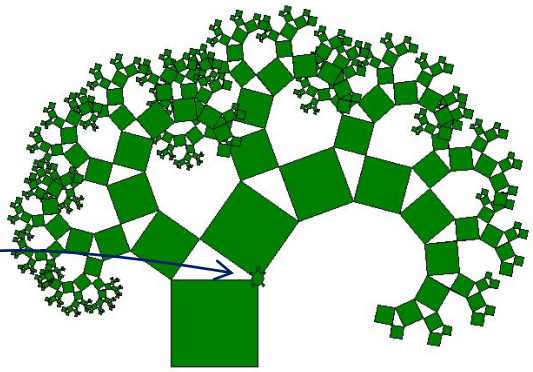
Pythagoras-Baum

Die Konstruktion eines Pythagoras-Baums basiert auf der Visualisierung des Satzes des Pythagoras: Die Fläche zweier *Quadrate* auf den kurzen Seiten (*Katheten*) eines rechtwinkligen Dreiecks entspricht der Fläche des *Quadrats* auf der langen Seite (*Hypothenuse*) des Dreiecks.



Jedes der beiden kleinen Kathetenquadrate wird im nächsten Schritt zum Hypothenusenquadrat, indem wieder zwei neue Kathetenquadrate im rechten Winkel zueinander oben draufgesetzt werden. Und jedes dieser beiden neuen Kathetenquadrate wird im nächsten Schritt… du ahnst es! Nach sieben Draufsetz-Runden sieht das Ergebnis so aus wie der Pythagoras-Baum im Bild rechts.



Von der Größe des spitzen *Winkels α* der rechtwinkligen Dreiecke hängt es ab, wie „schief" der Baum wird, also wie verschieden groß die beiden Kathetenquadrate werden: Bei 45° sind beide Winkel und beide Kathetenquadrate gleich groß, und der Baum wird symmetrisch. Probiere aus, was bei anderen Winkelgrößen passiert.
***Frage**: Warum müssen beide Winkel kleiner als 90° sein?*

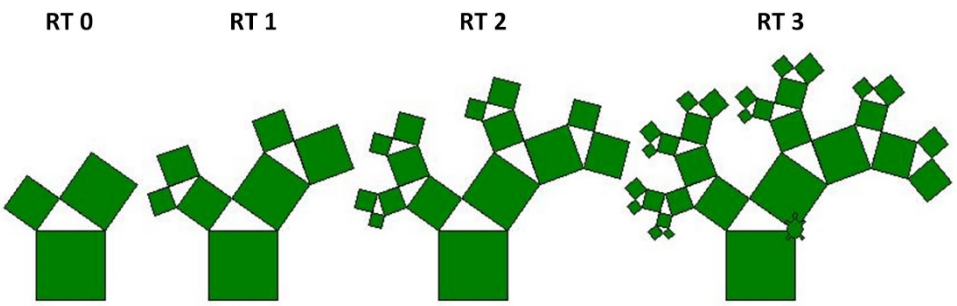
Abhängig vom Winkel *α* berechnest du die nächste linke Kathetenquadratbreite mit *BreiteL = Hypothenuse * sin(α)* und die rechte Kathetenquadratbreite mit *BreiteR = Hypothenuse * sin(90-α)*.

Hier ist das Rekursions-Prinzip im Pseudocode („RT" ist die Rekursionstiefe):

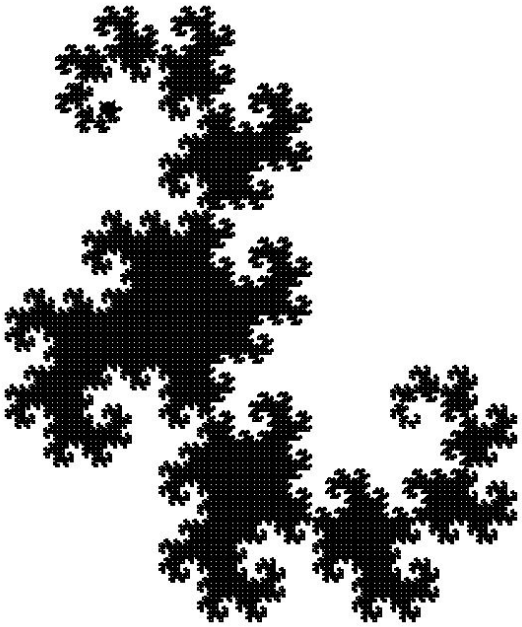
```
definition PythagorasBaum (Hypothenuse, α, RT):
  # 1. Male das Hypothenusenquadrat
   Quadrat (Hypothenuse)
 # 2. Berechne die Seiten der Kathetenquadrate
  BreiteL = Hypothenuse * sin(α)
  BreiteR = Hypothenuse * sin(90 -  α)
  # 3. Gehe zum Start für das rechte Quadrat
  [ …]
  # 4. Male die rechte Baumhälfte
  wenn RT > 0:
      PythygorasBaum (BreiteR, α, RT-1)
  sonst:   # fertig: letztes Quadrat rechts
      Quadrat (BreiteR)
  # 5. Gehe zum Start für das linke Quadrat
  [ …]
  # 6. Male die linke Baumhälfte
  wenn RT > 0:
      PythygorasBaum (BreiteL, α, RT-1)
  sonst:   # fertig: letztes Quadrat links
      Quadrat (BreiteL)
  # 7. Gehe zurück zum Hypothenusen-
Quadrat
  [ …]
```

Tipp: Am Start des Programms soll die Schildkröte nach oben schauen.

Rechts siehst du Pythagoras-Bäume verschiedener Rekursionstiefe:



Drachenkurve –
Der wachende Drache



Eine Drachenkurve entsteht, wenn du eine vertikal verlaufende Strecke in der Mitte rechtwinklig nach rechts abbiegst. Du erhältst zwei neue, halb so lange Strecken, eine vertikale und eine horizontale. Beide biegst du nun wieder mittig rechtwinklig um: die vertikale nach rechts (im Uhrzeigersinn) und die horizontale nach links (gegen Uhrzeigersinn). Mit dem Ergebnis machst du mit jeder halbierten Strecke das gleiche. Und so weiter. Nach 15 kompletten Faltvorgängen sieht das Ergebnis so aus wie der „Drache“ im Bild rechts.

Hier ist das Rekursions-Prinzip im Pseudocode („RT“ ist die Rekursionstiefe):

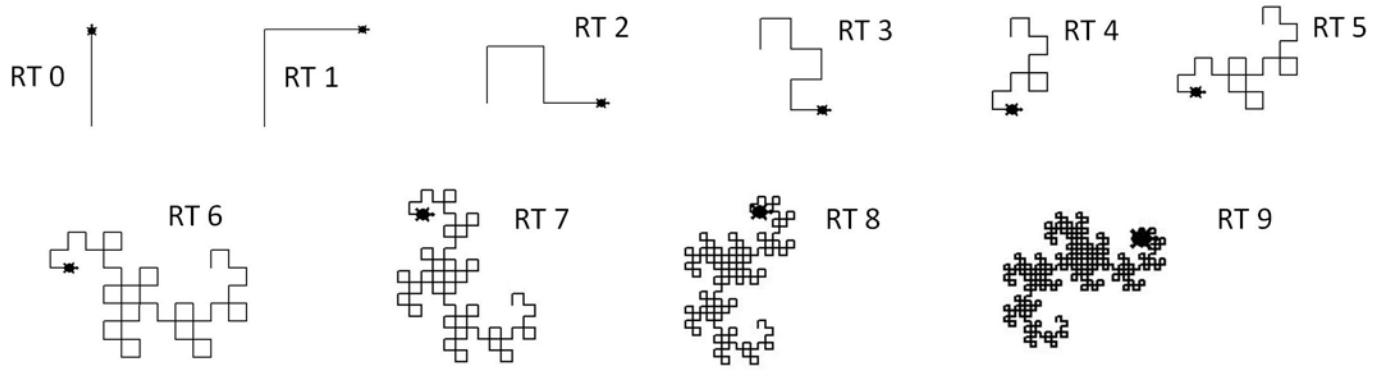
```
definition DrachenKurve (Strecke, RT):
  wenn (RT == 0):
    gehe (Strecke)                          # Rekursionstiefe 0: Male einfach nur eine Strecke
  sonst:
    StreckeVertikal (Strecke/2, RT-1)       # Male die erste Streckenhälfte (ggf. mit Knicken)
    rechts (90°)                            # Rechtsknick nach der ersten (vertikalen) Streckenhälfte
    StreckeHorizontal (Strecke/2, RT-1)     # Male die zweite Streckenhälfte (ggf. mit Knicken)
```

```
definition StreckeVertikal (Strecke, RT):
  wenn (RT == 0):
    gehe (Strecke)   # keine weiteren Knicke
  sonst:
    StreckeVertikal (Strecke/2, RT-1)
    rechts (90°)        # ein Knick nach rechts
    StreckeHorizontal (Strecke/2, RT-1)
```

```
definition StreckeHorizontal (Strecke, RT):
  wenn (RT == 0):
    gehe (Strecke)  # keine weiteren Knicke
  sonst:
    StreckeVertikal (Strecke/2, RT-1)
    links (90°)         # ein Knick nach links
    StreckeHorizontal (Strecke/2, RT-1)
```

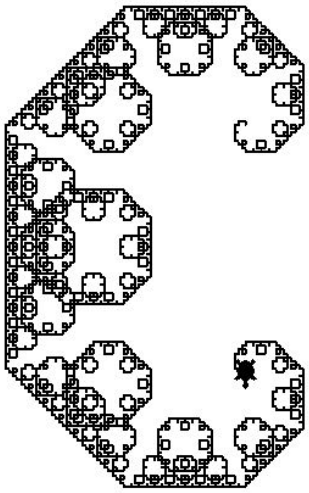
Tipp: Am Start des Programms soll die Schildkröte nach oben schauen.

Drachenkurven verschiedener Rekursionstiefe:



Levy-C-Kurve

Eine Levy-C-Kurve entsteht, wenn du eine vertikal verlaufende Strecke zu einem „C“ umformst, indem du oben und unten eine Querkante halber Streckenlänge anfügst. Anschließend wird für jede der vier Teilstrecken halber Streckenlänge (obere Querkante, obere vertikale Streckenhälfte, untere vertikale Streckenhälfte, untere Querkante) wieder das Gleiche gemacht: sie werden zu vier„C“-Schüsseln umgeformt, so dass wieder haufenweise neue, halb so lange Streckenteile entstehen. Auch diese werden im nächsten Schritt zu immer kleineren „C“-Schüsseln umgeformt. Nach sechs kompletten „C“-Umform-Vorgängen sieht das Ergebnis aus wie das sehr verschnörkelte „**C“** im Bild rechts.

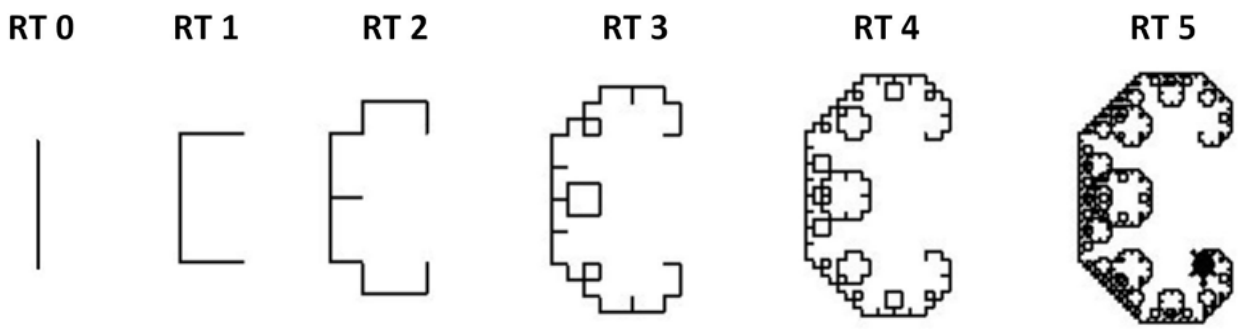


Hier ist das Rekursions-Prinzip im Pseudocode („RT“ ist die Rekursionstiefe):

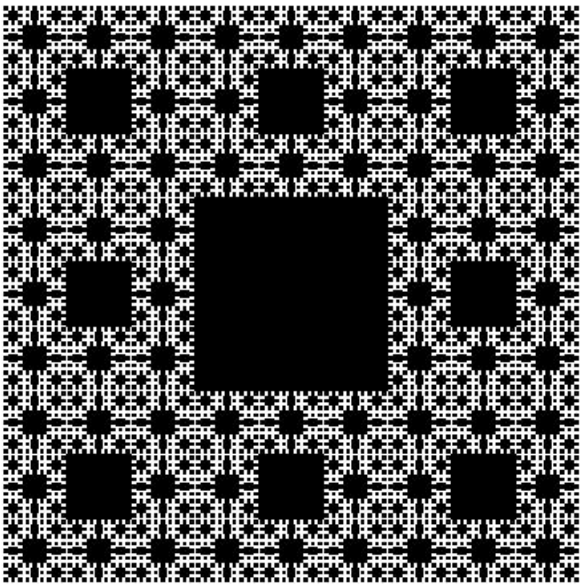
```
definition Levy-C-Kurve (Strecke, RT):
  wenn (RT == 0):
    gehe (Strecke)                    # Rekursionstiefe 0: Male einfach nur eine Strecke
  sonst:
    rechts (90°)                      # Ausrichten für die obere Querstrecke
    Levy-C-Kurve (Strecke/2, RT-1)    # Male die obere Querstrecke

    links (90°)                       # Ausrichten nach unten
    Levy-C-Kurve (Strecke/2, RT-1)    # Male die obere Hälfte vom Längs-Strich
    Levy-C-Kurve (Strecke/2, RT-1)    # Male die untere Hälfte vom Längs-Strich

    links (90°)                       # Ausrichten für die untere Querstrecke
    Levy-C-Kurve (Strecke/2, RT-1)    # Male die untere Querstrecke

    rechts (90°)                      # Ausrichten in Anfangsstellung (nach unten)
```

Tipp: Am Start des Programms soll die Schildkröte nach unten schauen, dann liegt das **C** nicht, sondern steht.

Drachenkurven verschiedener Rekursionstiefe:

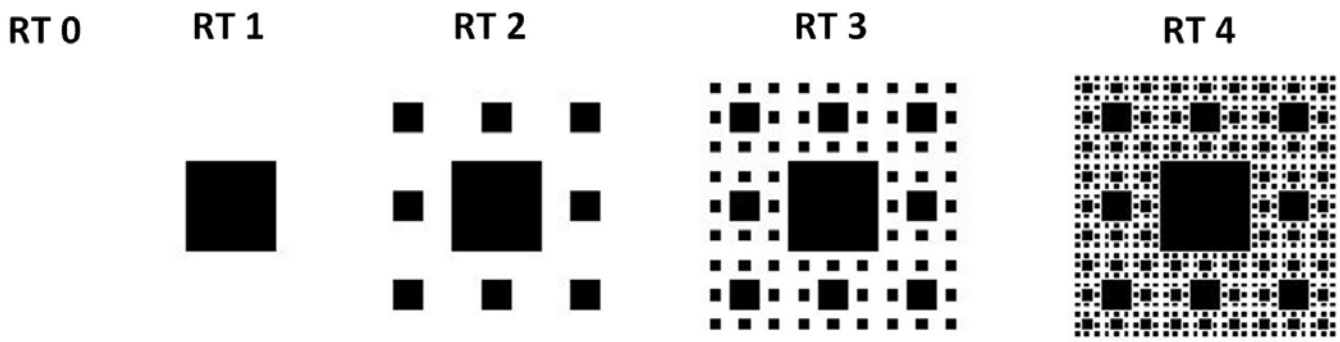


Sierpinski-Teppich



Ein Sierpinski-Teppich ist ein Quadrat. Im nächsten Level erhält dieses Quadrat acht kleine „Kinder“ -das sind auch Quadrate, die sich außen um das große Quadrat herum gruppieren: vier in den Ecken und vier mittig (oben, unten, links, rechts). Die Breite der Kinder-Quadrate ist ein Drittel der Breite vom großen Quadrat. Ein Level weiter erhält jedes der Kinder-Quadrate selber wieder Kinder-Quadrate. Und so weiter…
In Level 4 sieht das Ergebnis so aus wie der Teppich im Bild rechts. Natürlich kannst du die Quadrate auch mit einer anderen Farbe als Schwarz einfärben.

Hier ist das Rekursions-Prinzip im Pseudocode („RT“ ist die Rekursionstiefe):

```
definition SierpinskiTeppich (Breite, RT):
   # RT 0 bleibt einfach weiß
   wenn RT > 0:
      Quadrat ( Breite )              # 1. Male das große Quadrat

      X = neueXPos                    # 2. Berechne die Position vom ersten Kinder-Quadrat,
      Y = neueYPos                    #    (beginne z.B. links unten)
      goto (X, Y)

     SierpinskiTeppich (Breite/3, RT-1)     # 3. Male das erste der 8 Kinder-Quadrate

     #  4. Wiederhole jetzt die Schritte 2 und 3 noch 7 Mal und bewege dich dabei z. B. links um
           das große Quadrat herum
     [ … ]
     # 5. Wichtig! Am Ende gehe wieder zur Startposition
      goto (startX, startY)
```

Sierpinski-Teppich verschiedener Rekursionstiefe: