Rekursive Prozeduren, Teil 1

Verwendung des Turtle-Systems mit DrRacket

1. Bestimmen Sie ein Arbeitsverzeichnis und kopieren Sie die Datei racketturtle.rkt dort hinein.
2. Öffnen Sie eine neue Racket-Datei, in der Sie das Turtle-System verwenden möchten, und speichern Sie diese Datei im Arbeitsverzeichnis. Fügen Sie im Definitionsfenster die folgenden Zeilen ein:

```
#lang racket
(require "racketturtle.rkt")

(define width 800) ; Determines the window's size (turtles' playground)
(define height 800)

(define dc (start width height))
```

Nun sind Sie betriebsbereit. Die Fläche, in der sich die Turtle(s) bewegen, ist mit 800 mal 800 bereits vorgegeben.

Eine Turtle, wie etwa `harald`, wird mit folgender Anweisung (im Definitionsfenster!) instanziiert:

```
(define harald (new turtle%
                    [tname 'Harald]
                    [xpos 400][ypos 500][direction 90]
                    [tcolor "YellowGreen"]
                    [tdc dc]))
```

`harald` sitzt auf Position (400,500). Der Koordinatenursprung (0,0) befindet sich in der linken oberen Ecke. Die x-Koordinatenachse zeigt von dort nach rechts und die y-Achse zeigt nach unten.

Noch ist Harald nicht zu sehen. Das lässt sich (durch Eingabe im Interaktionsfenster!) ändern:

```
> (send harald show!)
```

Das spitzwinklig-gleichschenklige Dreieck gibt die Blick- und potenzielle Bewegungsrichtung von Harald an. Wir schicken ihn z.B. 50 Schritte vorwärts:

```
> (send harald forward! 50)
```

Eine rote Spur seiner Bewegung wird sichtbar. An deren Ende sitzt Harald und erwartet weitere Instruktionen.

Sämtliche Kommandos, die eine Turtle (wie Harald) versteht, finden sich in der folgenden Liste. Das Aufrufmuster ist `(send [turtle-name] [command])`.

Turtle-Kommandos

`say-your-name`	Die Turtle nennt ihren Namen.
`forward! n`	Die Turtle läuft `n` Schritte nach vorn.
`backward! n`	Die Turtle läuft `n` Schritte zurück.
`right! a`	Die Turtle dreht sich um `a` Grad nach rechts.
`left! a`	Die Turtle dreht sich um `a` Grad nach links.
`pen-up!`	Die Turtle hebt den Stift an. Die folgenden Bewegungen hinterlassen keine Spur.
`pen-down!`	Die Turtle senkt den Stift ab. Die folgenden Bewegungen hinterlassen eine Spur.
`pen-erase!`	Der Stift der Turtle verwandelt sich in einen Radierer.
`set-turtle-color! c`	Die Turtle ändert die Turtle-Farbe. Mögliche Werte: s. Racket color Database
`set-pen-color! c`	Die Turtle ändert die Stiftfarbe. Mögliche Werte: s. Racket color Database
`clone`	Die Turtle klont sich selbst.
`hide!`	Die Turtle versteckt sich.
`show!`	Die Turtle zeigt sich wieder (an aktueller Position).
`crow`	Die Turtle stößt einen Freudenschau (Glockenklang) aus.
`sleep n`	Die Turtle schläft `n` Millisekunden.

Machen Sie sich mit diesen Kommandos durch aktive Kommunikation mit Harald vertraut.

Eigene Turtle-Prozeduren

Im Folgenden werden eigene Prozeduren angegeben, die diese Grundkommandos verwenden und ganz bestimmte grafische Darstellungen erzeugen, beispielsweise ein Quadrat:

```
(define square
  (lambda (side turtle)
    (send turtle forward! side)
    (send turtle right! 90)
```

```
    (send turtle forward! side)
    (send turtle right! 90)
    (send turtle forward! side)
    (send turtle right! 90)
    (send turtle forward! side)
    (send turtle right! 90)))

> (square 100 harald)
```

Im Definitionstext erkennt man die vierfache Ausführung von

```
    (send turtle forward! side)
    (send turtle right! 90)
```

was zur Erzeugung eines Quadrats notwendig ist.

Um den Schreibaufwand (und damit die Fehlergefahr) zu verringern, setzen wir das Sprachelement `for` ein, das *Iterationen* ermöglicht.

```
(define square2
  (lambda (side turtle)
    (for ([n (in-range 4)])
      (send turtle forward! side)
      (send turtle right! 90))))

> (square2 100 harald)
```

Aufgaben

1. Schreiben Sie eine Prozedur, die ein Rechteck mit den Seitenlängen `a` und `b` mit einer gegebenen Turtle zeichnet.
2. Schreiben Sie eine Prozedur, mit deren Hilfe das Haus vom Nikolaus gezeichnet werden kann. Die Prozedur soll als Argumente die Turtle sowie die Länge der Grundseite des Hauses übernehmen.
3. Schreiben Sie eine rekursive Prozedur `kreisbogen`, die eine Turtle und die Bogenlänge nimmt und einen dementsprechenden Kreisbogen zeichnet.

   Hinweis: Ein Turtle-Kreis ist ein 360-Eck. Jeder der 360 Turtle-Schritte hat die Länge 1. Nach jedem Schritt dreht sich die Turtle um 1 Grad nach rechts (oder links). Der Umfang dieses Kreises beträgt offenbar 360. (Ganz nebenbei: Wie groß ist dann eigentlich der Radius?)

4. Erproben Sie die rekursive Prozedur `tree` aus der Vorlesung. Machen Sie sich die rekursive Idee durch Beschreibung des grafischen Resultats vollständig klar.

```
(define tree
  (lambda (side turtle)
    (if (< side 5)
        (send turtle crow)
        (begin
          (send turtle forward! side)
          (send turtle left! 45)
          (send turtle sleep 10)
          (tree (/ side 2) turtle)
          (send turtle right! 90)
          (send turtle sleep 10)
          (tree (/ side 2) turtle)
          (send turtle left! 45)
          (send turtle backward! side)))))
```

```
> (tree 100 harald)
```

Vor diesem Aufruf kann man die turtle bei angehobenem Stift rückwärts bewegen und dann den Stift absenken lassen. Auf diese Weise gewinnt man auf der Zeichenfläche mehr Platz für die Zeichnung.

Hinweis: Je nach Geschwindigkeit Ihres Rechners sollten Sie die Verzögerung der grafischen Darstellung durch das konkrete Argument (Zahlenwert in Millisekunden) von `sleep-for-a-while` anpassen.

5. Lassen Sie sich von den schönen Rosetten (s.u.) inspirieren und entwerfen Sie ähnliche. Beginnen Sie stets mit der Planung und verbalen Beschreibung der zu zeichnenden Figuren. Die Implementierung erfolgt mit rekursiven Turtle-Prozeduren.