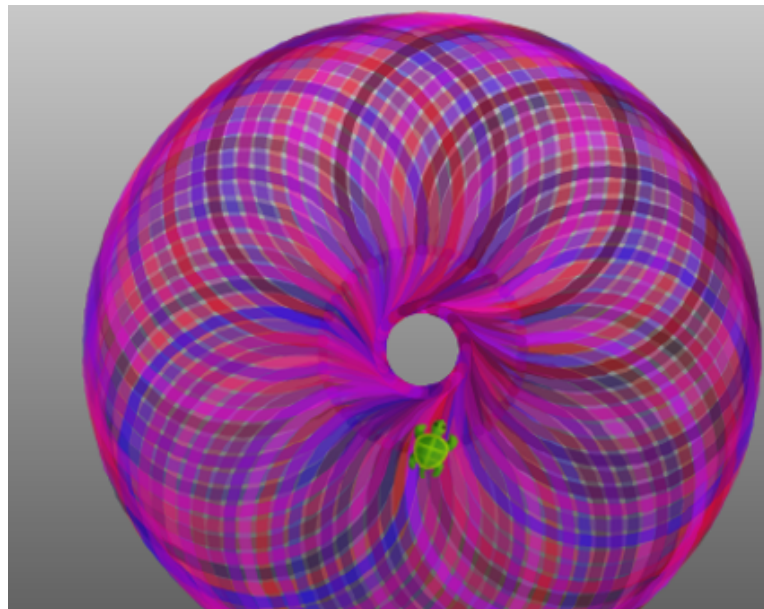


Aufgaben mit Kojo

Editor: Björn Regnell
www.lth.se/code



Aufgaben mit Kojo

Version: 15. April 2016



License: Creative Commons *Attribution-NonCommercial-ShareAlike 4.0 International* CC BY-NC-SA 4.0

Editor: Björn Regnell

Translation to German: Simone Strippgen, Christoph Knabe

Contributors: Björn Regnell, Lalit Pant, Sandra Nilsson, Maja Johansson, Simone Strippgen, Christoph Knabe

© Björn Regnell, Lund University, 2015

<http://lth.se/code>

Inhalt

Über Kojo	1	Zeichne ein Vieleck	15	Nutze eine Solange-Schleife	31
Dein erstes Programm	2	Zeichne viele Vielecke	16	Zahlenraten	32
Zeichne ein Quadrat	3	Werte und Ausdrücke	17	Übe Multiplikation	33
Zeichne eine Treppe	4	Gebe einem Wert einen Namen mit <code>val</code>	18	Speichere Tiere in einem Vektor	34
Nutze eine Schleife	5	Zufallszahlen	19	Übe Vokabeln	35
Zeichne eine Figur	6	Mische Deine eigenen Farben	20	Hauptstadt-Spiel	36
Wie schnell ist Dein Computer?	7	Probiere die Farbauswahl	21	Erstelle eine Stoppuhr mit <code>object</code>	37
Verfolge Dein Programm	8	Zeichne Zufallskreise	22	Simuliere eine Ampel	38
Erstelle Deine eigene Funktion mit <code>def</code>	9	Zeichne eine Blume	23	Steuere die Kröte mit der Tastatur	39
Stapel Quadrate	10	Erstelle eine Variable mit <code>var</code>	24	Steuere die Kröte mit der Maus	40
Schreibe eine Stapelfunktion	11	Zeichne viele Blumen	25	Erstelle Dein eigenes Bankkonto	41
Erstelle ein Gitter	12	Gebe der Kröte ein neues Kostüm	26	Erstelle viele Objekte mit <code>class</code>	42
Quadrat mit Parameter	13	Erstelle weitere Kröten mit <code>new</code>	27	Sprich mit Deinem Computer	43
Zeichne einen Quadratkopf	14	Krötenrennen	28	Verändere das Tischtennis-Spiel	44
		Alternative mit <code>if</code>	29		
		Reagiere auf das, was der Nutzer macht	30		

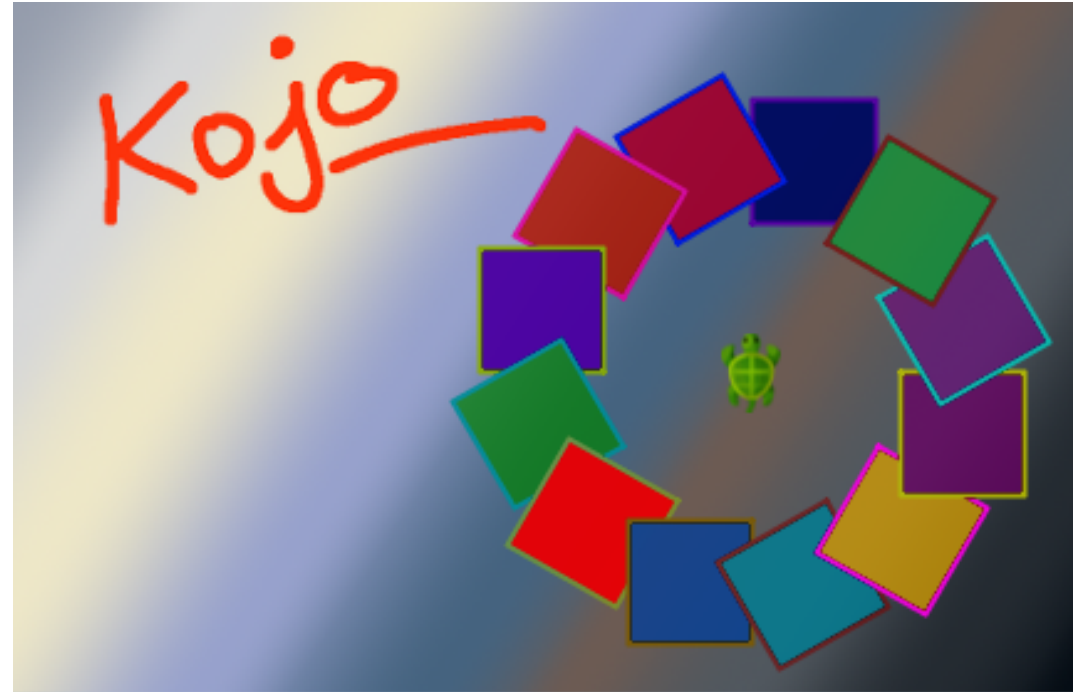
Über Kojo

Was ist Kojo?

Kojo ist eine App, die Dir beibringt, wie man programmiert. Mit Kojo benutzt Du die moderne und leistungsfähige Programmiersprache **Scala**. Kojo gibt es kostenlos und auf Deutsch. Kojo läuft auf Linux, Windows und Mac OS X.

Wo kannst Du Kojo finden?

Lade Kojo hier:
www.kogics.net/kojo-download
Lies hier mehr darüber:
lth.se/programmera



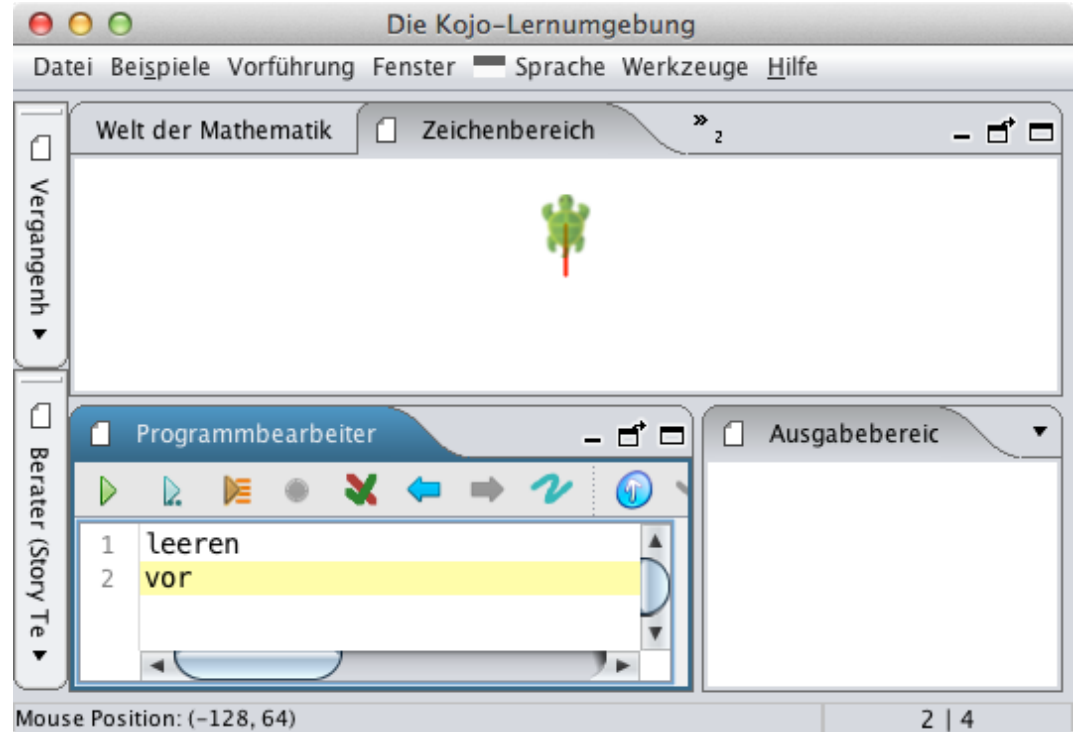
Dein erstes Programm

Aufgabe:

Schreibe diese Worte in Kojos Programmierer:

leeren
vor

Klicke auf das grüne Ausführen-Symbol
um das Programm zu starten.



Zeichne ein Quadrat

leeren
vor
rechts

Bei Eingabe von links oder rechts dreht sich die Kröte.

Aufgabe:

Erweitere das Programm so, dass ein Quadrat gezeichnet wird.



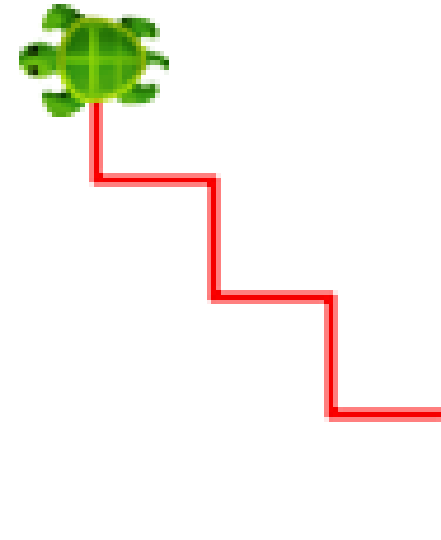
Zeichne eine Treppe

leeren
vor; links
vor; rechts

Mehrere Befehle in einer Zeile müssen durch ein Semikolon ; getrennt werden.

Aufgabe:

Erweitere das Programm so, dass eine Treppe gezeichnet wird.



Nutze eine Schleife

leeren
schleife(4){ vor; rechts }



Aufgabe:

- Was passiert, wenn Du statt 4 die Zahl 100 eingibst?
- Zeichne eine Treppe mit 100 Stufen.

Zeichne eine Figur

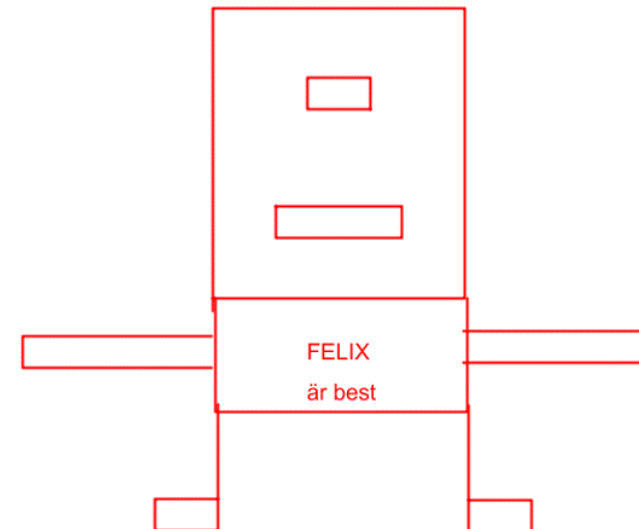
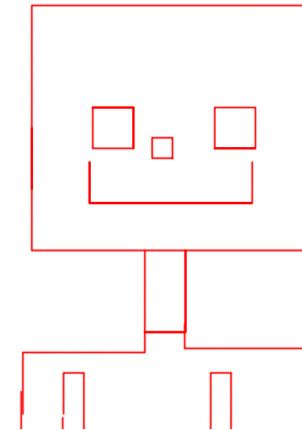
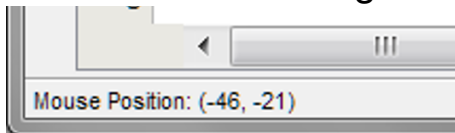
Aufgabe:

Zeichne eine einfache Figur.

Tipps:

```
springen  
links(180)  
vor(300)  
springen(100)  
springen(25,-28)  
schreiben("FELIX ist der Beste")  
stiftfarbe(lila)  
füllfarbe(grün)
```

Die Position der Maus im Zeichenbereich kannst Du links unter dem Programmbearbeiter ablesen:



Wie schnell ist Dein Computer?

Der erste Computer hieß **ENIAC** und konnte in einer Sekunde bis 5000 zählen.
In Kojo gibt es eine Funktion `zählzeitStoppen` die misst, wie schnell Dein Computer zählen kann.
Wenn ich `zählzeitStoppen(5000)` ausführe, dann wird die Zeit ausgegeben, die mein Computer für das Zählen braucht:

```
*** Zählt von 1 bis ... 5000 *** FERTIG!  
Es dauerte 0.32 Millisekunden.
```

Aufgabe:

- Gib ein `zählzeitStoppen(5000)` und prüfe, ob Dein Computer schneller ist als meiner.
- Wie lange braucht Dein Computer, um bis eine Million zu zählen?
- Wie weit kann Dein Rechner in einer Sekunde zählen?

Verfolge Dein Programm

Aufgabe:

- Schreibe ein Programm, das eine Stufe zeichnet.
- Klicke auf das orange Verfolgen-Symbol.
- Klicke auf einen der Aufrufe: CALL vor (). Was passiert im Zeichenbereich?
- Im Programmbeurbeiter wird der zugehörige Befehl blau markiert. Deaktiviere die Markierung, indem Du neben die Markierung klickst.
- Füge dem Programm weitere Befehle hinzu und probiere aus, was passiert, wenn Du sie verfolgst
- Schließe das Fenster *Programmschritte verfolgen* wenn Du fertig bist.



Erstelle Deine eigene Funktion mit **def**

Mit **def** kannst Du einer eigenen *Funktion* einen Namen geben.

```
def quadrat = schleife(4){ vor; rechts }
```

leeren

```
quadrat //Ruft die Quadrat-Funktion auf.
```

springen

```
quadrat
```

Aufgabe:

- Ändere die Farbe der Quadrate.
- Erstelle mehr Quadrate.

Tipps:

```
füllfarbe(grün); stiftfarbe(lila)
```

Stapel Quadrate

Aufgabe:

Erstelle einen Stapel von 10 Quadraten.

Tipps:

```
def quadrat = schleife(4){ vor; rechts }
```

```
leeren; langsam(100)  
schleife(10){ ??? }
```



Schreibe eine Stapelfunktion

Aufgabe:

Schreibe eine Funktion mit dem Namen `stapel`, die einen Stapel aus 10 Quadraten zeichnet.

Tipps:

```
def quadrat = schleife(4){ vor; rechts }  
def stapel = ???
```

```
leeren; langsam(100)  
stapel
```



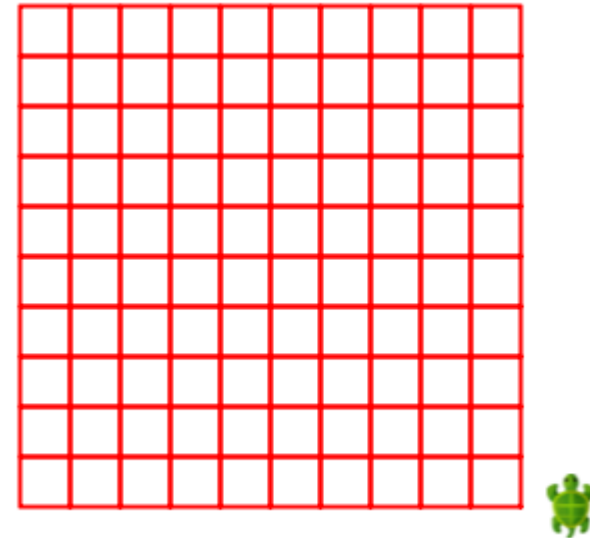
Erstelle ein Gitter

Aufgabe:

Erstelle ein Gitter aus $10 * 10$ Quadraten.

Tipps:

- Verwende hierfür Deine Stapelfunktion von eben.
- Du kannst eine ganze Stapelhöhe zurück springen mit `springen(-10 * 25)`
- Du kannst dann an die richtige Stelle springen mit `rechts; springen; links`



Quadrat mit Parameter

Aufgabe:

Zeichne verschiedene Quadrate.

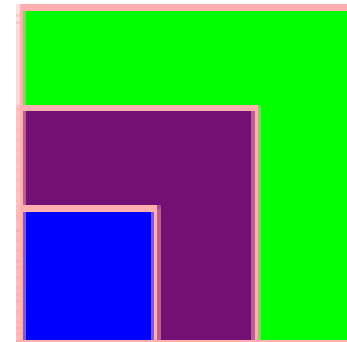
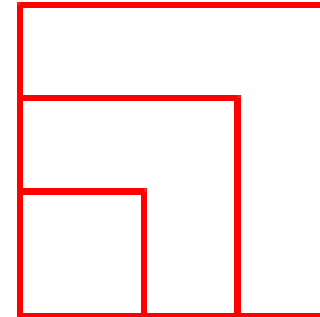
Tipps:

Gib Deiner Quadrat-Funktion einen *Parameter* mit dem Namen *seitenlänge* und Typ *Ganzzahl*:

```
def quadrat(seitenlänge : Ganzzahl) =  
  schleife(4){ vor(seitenlänge); rechts }
```

```
leeren; langsam(100); unsichtbar  
quadrat(100)  
quadrat(70)  
quadrat(40)
```

Du kannst die Farbe ändern mit:
füllfarbe(blau); stiftfarbe(rosa)



Zeichne einen Quadratkopf

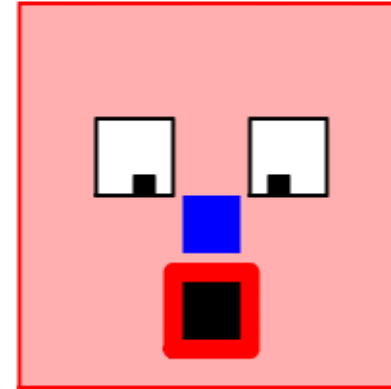
Aufgabe:

Zeichne einen Kopf aus verschiedenen Quadraten.

Tipps:

```
def quadrat(x: Ganzzahl, y: Ganzzahl, seitenlänge: Ganzzahl) = {  
    springen(x, y)  
    schleife(4) { vor(seitenlänge); rechts }  
}  
def kopf(x: Ganzzahl, y: Ganzzahl) = { füllfarbe(rosa); stiftfarbe(rot); quadrat(x, y, 200) }  
def auge(x: Ganzzahl, y: Ganzzahl) = { füllfarbe(weiß); stiftfarbe(schwarz); quadrat(x, y, 40) }  
def pupille(x: Ganzzahl, y: Ganzzahl) = { füllfarbe(schwarz); stiftfarbe(schwarz); quadrat(x, y, 10) }  
def nase(x: Ganzzahl, y: Ganzzahl) = { füllfarbe(blau); stiftfarbe(durchsichtig); quadrat(x, y, 30) }  
def mund(x: Ganzzahl, y: Ganzzahl) = { stiftbreite(10); füllfarbe(schwarz);  
    stiftfarbe(rot); quadrat(x, y, 40) }
```

```
leeren; langsam(20); unsichtbar  
kopf(0, 0)  
auge(40, 100); pupille(60, 100)  
???
```



Zeichne ein Vieleck

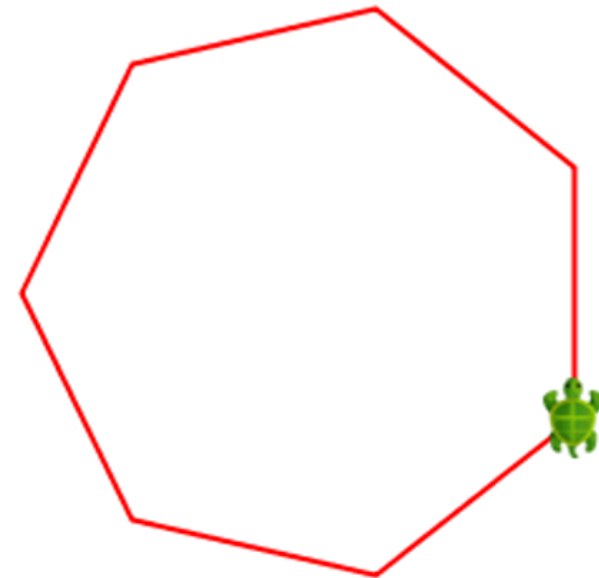
Aufgabe:

- Probiere das Programm unten aus. Zeichne verschiedene Vielecke.
- Füge einen Parameter seitenlänge hinzu und zeichne verschieden große Vielecke.
- Wie groß muss n sein, damit ein Kreis entsteht?

Tipps:

```
def vieleck(n: Ganzzahl) = schleife(n){  
  vor(100)  
  links(360.0/n)  
}
```

```
leeren; langsam(100)  
vieleck(7)
```

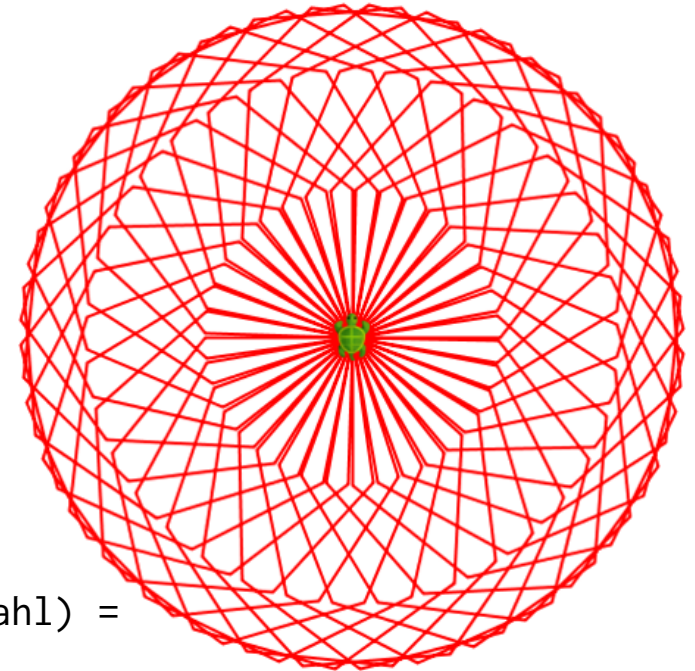


Zeichne viele Vielecke

Aufgabe:

- Probiere das Programm unten aus.
- Verändere die Anzahl der Seiten und den Winkel.
- Fülle die Vielecke mit Farbe.

```
def vieleck(n: Ganzzahl, seitenlänge: Ganzzahl) =  
  schleife(n){  
    vor(seitenlänge)  
    links(360.0/n)  
  }  
def drehen(n: Ganzzahl, winkel: Ganzzahl, seitenlänge: Ganzzahl) =  
  schleife(360/winkel){  
    vieleck(n, seitenlänge)  
    links(winkel)  
  }  
  
leeren; langsam(5)  
drehen(7, 10, 100)
```



Werte und Ausdrücke

Aufgabe:

- Schreibe `1 + 1` und klicke die blaue Ausführen-Taste. Dann erstellt Kojo einen grünen Kommentar.
- Der Kommentar sagt, dass der Wert des Ausdrucks `1 + 1` gleich 2 ist und dass der Typ `Int` ist. `Int` ist eine Abkürzung für Integer, auf Deutsch: Ganzzahl.
- Führe weitere Berechnungen durch. Welcher Wert und Typ wird ausgegeben?

`5 * 5`

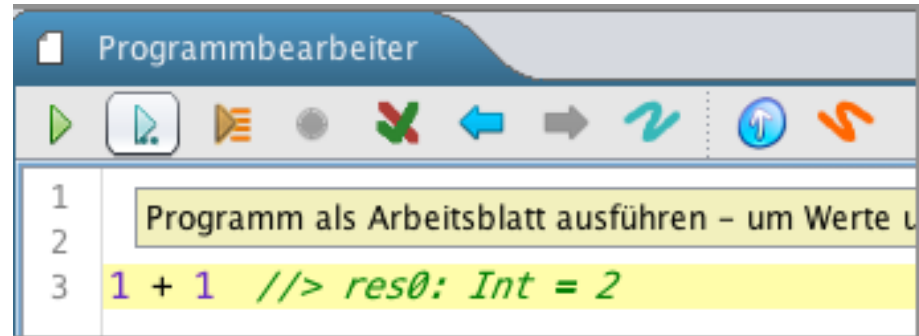
`10 + 2 * 5`

`"Auf" + " Wieder" + "sehen"`

`5 / 2`

`5 / 2.0`

`5 % 2`



Tipps:

- Steht `/` zwischen ganzen Zahlen, so wird eine ganzzahlige Division ausgeführt. Das heißt, die Nachkommastellen entfallen. Für eine Division mit Nachkommastellen muss mindestens eine Zahl eine Bruchzahl sein.
- Mit `%` erhältst Du den Rest einer ganzzahligen Division.

Gebe einem Wert einen Namen mit **val**

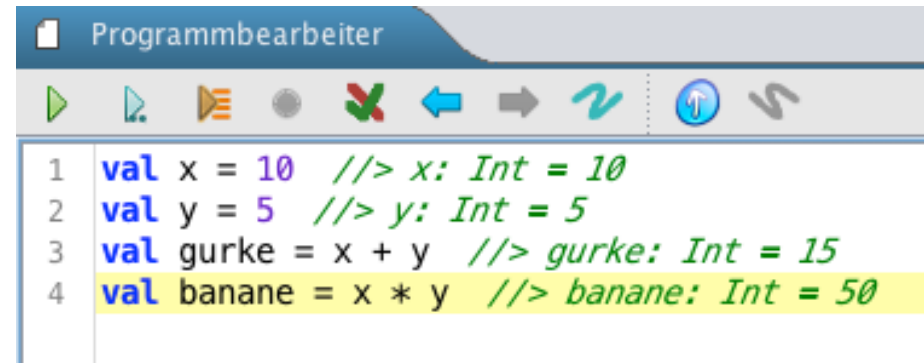
Aufgabe:

Mit **val** kannst Du einen Namen mit einem Wert koppeln. Der Name kann dann anstelle des Wertes verwendet werden. Probiere das folgende Programm aus. Was schreibt die Kröte?

```
val x = 10
val y = 5
val gurke = x + y
val banane = x * y
```

leeren

```
vor; schreiben(banane)
vor; schreiben(gurke)
vor; schreiben(y)
vor; schreiben(x)
```



```
1 val x = 10 //> x: Int = 10
2 val y = 5 //> y: Int = 5
3 val gurke = x + y //> gurke: Int = 15
4 val banane = x * y //> banane: Int = 50
```

Zufallszahlen

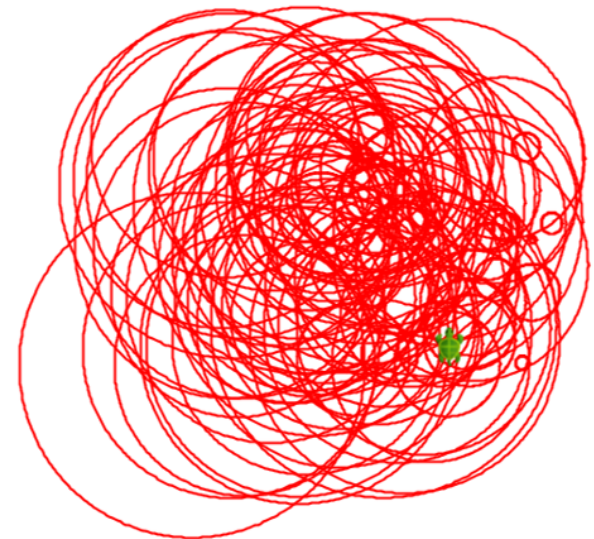
Aufgabe:

- Führe das Programm unten mehrmals aus. Was passiert?
- Was ist der kleinste und der größte mögliche Wert des Radius r ?
- Ändere r zu einer Zufallszahl zwischen 3 und 200.
- Zeichne 100 Kreise mit zufälligem Radius und zufälliger Position wie im Bild rechts.

// zufall(90) liefert eine Zufallszahl zwischen 0 und 99:

```
val r = zufall(90) + 10
```

```
leeren; langsam(10); unsichtbar  
schreiben("Radius = " + r)  
kreis(r)
```



Mische Deine eigenen Farben

- Mit Color kannst Du Deine eigenen Farben erstellen, z.B. `Color(0, 70, 0)`
- Die drei Parameter geben den Farbanteil von *rot*, *grün* und *blau* an.
- Du kannst auch einen vierten Parameter für die *Deckkraft* angeben.
- Alle Parameterwerte müssen zwischen 0 und 255 liegen.

Aufgabe:

Probiere das folgende Programm. Ändere die Deckkraft.

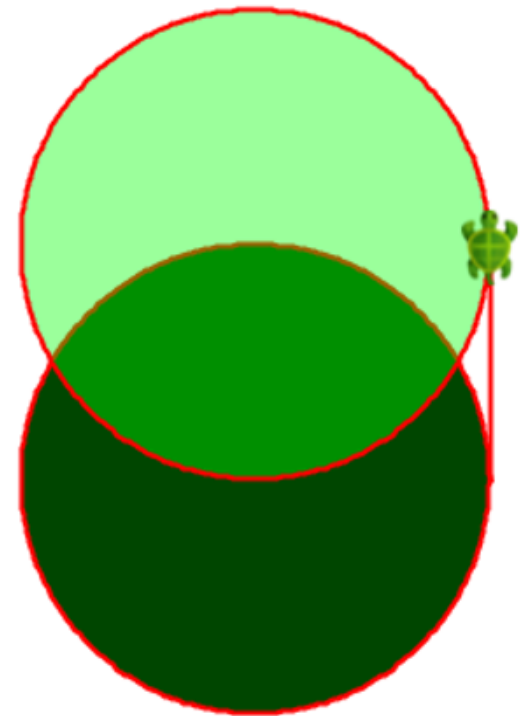
```
leeren; langsam(100)
```

```
val olivgrün = Color(0,70,0)
```

```
val pistazie = Color(0,255,0,100)
```

```
füllfarbe(olivgrün); kreis(100)
```

```
füllfarbe(pistazie); vor(100); kreis(100)
```



Probiere die Farbauswahl

Aufgabe:

- Mache einen Rechtsklick im Programmbeurbeiter und gehe auf Farbe auswählen...
- Unter **RGB** kannst Du eine neue RGB-Farbe mischen.
- Drücke OK und schaue in den Ausgabebereich. Hier werden die RGB-Werte für Rot, Grün und Blau angezeigt.
- Du kannst diese Werte verwenden, um mit der ausgewählten Farbe zu zeichnen
`stiftfarbe(Color(218, 103, 67)).`



Zeichne Zufallskreise

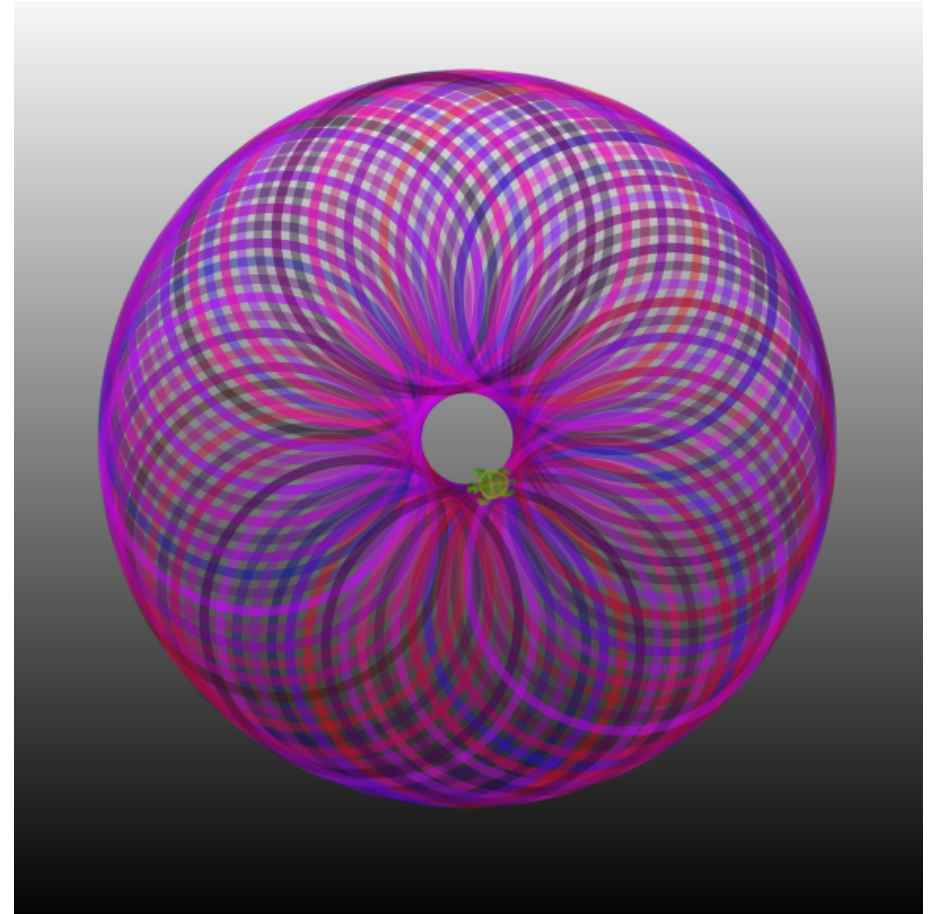
```
def zufällig = zufall(256)
def zufallsfarbe =
    Color(zufällig,10, zufällig,100)

leeren; langsam(5)
grundfarbeU0(schwarz,weiß)
stiftbreite(6)

schleife(100) {
    stiftfarbe(zufallsfarbe)
    kreis(100)
    springen(20)
    rechts(35)
}
```

Aufgabe:

Probiere verschiedene zufällige Stift- und Hintergrund-Farben aus.

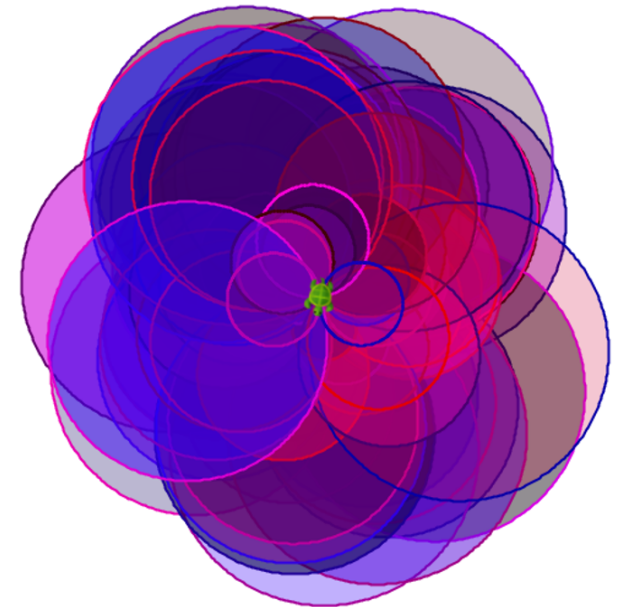


Zeichne eine Blume

Aufgabe:

Das folgende Programm zeichnet 100 Kreise mit zufälliger Farbe, zufälligem Radius und zufälliger Ausrichtung. Probiere verschiedene Zufallszahlen-Grenzen und versuche zu erklären, was hierbei passiert.

```
leeren(); langsam(5)
stiftbreite(2)
schleife(100){
  stiftfarbe(Color(zufall(256),0,zufall(256)))
  füllfarbe(Color(zufall(256),0,zufall(256),zufall(100)+50))
  links(zufall(360))
  kreis(zufall(30)*4+10)
}
```



Erstelle eine Variable mit **var**

Mit **var** kannst Du einen Namen mit einem Wert koppeln.
Einer Variablen kann man jederzeit einen neuen Wert zuweisen:

```
var gurke = 1  
gurke = 1 + 1 //erst wird 1 + 1 ausgerechnet, gurke bekommt dann den Wert 2
```

Aufgabe:

Probiere das Programm aus. Was schreibt die Kröte?

```
var i = 0  
  
leeren  
schleife(10){  
  i = i + 1  
  vor; schreiben(i)  
}
```

Tipps:

- Die Zuweisung `i = i + 1` weist `i` einen neuen Wert zu, der sich aus dem *alten* Wert von `i` plus 1 berechnet.

Zeichne viele Blumen

Aufgabe:

- Schreibe eine Funktion mit dem Namen `blume`, die eine Blüte zeichnet und einen Stiel mit einem grünen Blatt, der in der Mitte der Blüte beginnt.
- Zeichne 5 Blumen nebeneinander.

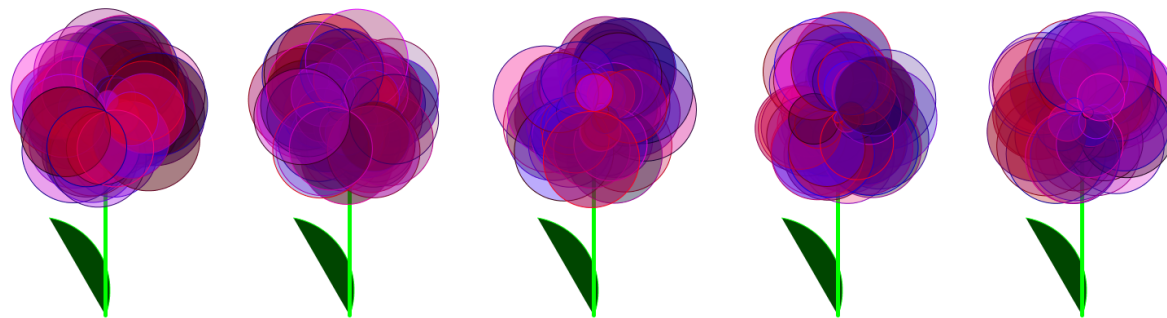
Tipps:

Du kannst die Blätter zeichnen mit `bogen(radius, winkel)`.

Gib der Funktion `blume` zwei Parameter `x` und `y` und benutze `springen(x,y)`

Du kannst die Schleife fünfmal ausführen und die Position wie folgt ausrechnen:

```
var i = 0
schleife(5){
  blume(600*i,0)
  i = i + 1
}
```



Gebe der Kröte ein neues Kostüm

Aufgabe:

Lade die Mediendateien von Kojos Webseite: www.kogics.net/kojo-download#media

- Entpacke die Datei `scratch-media.zip` und finde das Krabbenbild `crab1-b.png` im Ordner `Media/Costumes/Animals`
- Speichere die Datei `crab1-b.png` in dem Ordner, in dem auch das Programm liegt.
- Ziehe der Kröte ein Krabbenkostüm an:

```
leeren  
kostüm("crab1-b.png")  
langsam(2000)  
vor(1000)
```



Tipps:

- Du kannst auch eigene Bilder vom Typ `.png` oder `.jpg` verwenden.
- Wenn Du das Bild in einem anderen Ordner speichern wilt, musst Du den Dateipfad angeben, z.B. `kostüm("~/Kujo/Media/Costumes/Animals/crab1-b.png")`. Das Zeichen `~` steht für Dein Home-Verzeichnis.

Erstelle weitere Kröten mit **new**

Du kannst viele neue Kröten mit **new** hinzufügen:

leeren

```
val k1 = new Kröte(100,100) //die neue Kröte k1 startet auf der Position (100, 100)
```

```
val k2 = new Kröte(100, 50) //die neue Kröte k2 startet auf der Position (100, 50)
```

```
k1.vor(100)
```

```
k2.rück(100) //Kröte k2 nach unten
```



Aufgabe:

- Erzeuge drei Kröten übereinander.
- Alle drei sollen nach links schauen.



Tipps:

- k1 und k2 sind die *Namen* der neuen Kröten. Du kannst die Namen ändern, wenn Du willst.
- Mit k1 gefolgt von einem Punkt kannst Du der Kröte k1 einen Befehl geben: k1.links
- Mit dem Befehl unsichtbar kannst Du eine Kröte verbergen.

Krötenrennen

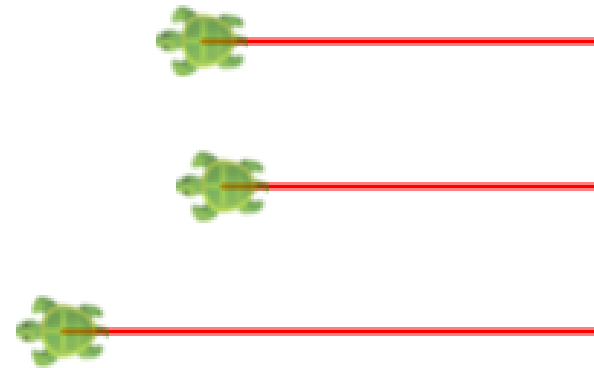
Mit Hilfe von Zufallszahlen kannst Du die Kröten um die Wette rennen lassen.

Aufgabe:

- Lass die drei Kröten laufen.
- Wenn alle Kröten 10 mal nach vorn laufen, welche Kröte ist dann die Erste?

Tipps:

- Mit `k1.vor(zufall(100) + 1)` bewegt sich k1 zwischen 1 und 100 Schritten weit.



Alternative mit **if**

Mit einer **if**-Anweisung kann man den Computer aufgrund einer Bedingung zwischen zwei Alternativen wählen lassen.

leeren; unsichtbar

```
if (true) schreiben("wahr") else schreiben("falsch")
```

Aufgabe:

- Ändere die Bedingung **true** in **false** und prüfe, was die Kröte schreibt.
- Ändere die Bedingung in $2 > 1$ und prüfe, was die Kröte schreibt.
- Ändere die Bedingung in $2 < 1$ und prüfe, was die Kröte schreibt.
- Erkläre wie eine **if**-Anweisung funktioniert.

Tipps:

- Hinter **if** kommt immer eine Bedingung in Klammern.
- Wenn die Bedingung nach dem **if** gleich **true** ist, wird der Befehl hinter der Bedingung ausgeführt.
- Wenn die Bedingung nach dem **if** gleich **false** ist, wird der Befehl hinter **else** ausgeführt.

Reagiere auf das, was der Nutzer macht

```
ausgabeLeeren; setOutputTextFontSize(35)
val passwort = "gurke"
val frage     = "Wie lautet mein Passwort?"
val richtig   = "Der Safe ist offen!"
val fehler    = "Du kommst nicht hinein!"
val antwort = einlesen(frage) //Warten auf die Antwort des Nutzers
val meldung = if (antwort == passwort) richtig else fehler
ausgeben(meldung)
```

Aufgabe:

- Probiere das Programm aus. Du musst im Ausgabebereich unten etwas eingeben und mit der Eingabetaste bestätigen. Erkläre, was passiert.
- Ändere das Passwort. Was wird ausgegeben, wenn man es richtig und wenn man es falsch eingibt?
- Frage auch nach einem Benutzernamen und gib den Namen aus.

Nutze eine Solange-Schleife

Mit `schleifeSolange` wiederholt der Computer Anweisungen so lange, wie eine Bedingung wahr ist.

```
leeren; unsichtbar; langsam(250); ausgabeLeeren
```

```
var x = 200
```

```
schleifeSolange (x > 0) { //prüfe die Bedingung vor jeder Wiederholung
```

```
    vor(x); rechts
```

```
    schreiben(x)
```

```
    x = x - 12
```

```
}
```

```
ausgeben("x ist jetzt: " + x)
```

Aufgabe:

- Was wird im Ausgabebereich ausgegeben? Warum?
- Starte das Programm mit dem orangen Verfolgen-Symbol und prüfe jeden Schritt.
- Ziehe statt 12 den Wert 20 von der Variablen x ab. Erkläre, was passiert.

Zahlenraten

```
val geheimnis = zufall(100)+1
var antwort = einlesen("Rate eine Zahl zwischen 1 und 100! ")
var überspringen = true

schleifeSolange (überspringen) {
    if (antwort.toInt < geheimnis)
        antwort = einlesen(antwort + " ist zu KLEIN, rate erneut!")
    else if (antwort.toInt > geheimnis)
        antwort = einlesen(antwort + " ist zu GROSS, rate erneut!")
    else if (antwort.toInt == geheimnis)
        überspringen = false
}
ausgeben(geheimnis + " ist die RICHTIGE Antwort!")
```

Aufgabe:

Führe eine Variable `var anzahlVersuche = 0` ein und stelle sicher, dass am Ende ausgegeben wird: Richtige Antwort! Du hast es mit 5 Versuchen geschafft

Übe Multiplikation

```
var anzahlRichtig = 0
val startZeit = System.currentTimeMillis / 1000
schleife(12) {
    val zahl1 = zufall(12)+1
    val zahl2 = zufall(12)+1
    val antwort = einlesen("Was ist " + zahl1 + "*" + zahl2 + "?")
    if (antwort == (zahl1 * zahl2).toString) {
        ausgeben("Richtig!")
        anzahlRichtig = anzahlRichtig + 1
    }
    else ausgeben("Falsch. Die richtige Antwort ist " + (zahl1 * zahl2))
}
val stoppZeit = System.currentTimeMillis / 1000
val sekunde = stoppZeit - startZeit
ausgeben("Du hast " + anzahlRichtig + " richtige Antworten in " + sekunde + " Sekunden.")
```

Aufgabe:

Ändere das Programm so, dass man nur noch die Multiplikation mit 8 und 9 üben muss.

Speichere Tiere in einem Vektor

```
var tiere = Vector("Elch", "Kuh", "Kaninchen", "Möve") //Variable tiere enthält einen Vektor mit 4 Tieren
ausgeben("Das erste Tier im Vektor ist: " + tiere(0)) //Nummerierung der Werte im Vektor beginnt bei 0
ausgeben("Ein anderes Tier im Vektor ist: " + tiere(1))
ausgeben("Der Vektor enthält so viele Tiere: " + tiere.size)
ausgeben("Das letzte Tier im Vektor ist: " + tiere(tiere.size-1))

val s = zufall(tiere.size) //berechnet eine Zufallszahl zwischen 0 und der Anzahl der Tiere minus 1
ausgeben("Ein zufälliges Tier: " + tiere(s))

tiere = tiere :+ "Kamel" //fügt das Tier am Ende des Vektors ein
tiere = "Dromedar" +: tiere //fügt das Tier am Anfang des Vektors ein
tiere = tiere.updated(2, "Nilpferd") //Ändert das dritte Tier (Nummer 2 im Vektor)
ausgeben("Alle Tiere rückwärts:")
tiere.foreach{ x => ausgeben(x.reverse) } //für alle x im Vektor: gebe x rückwärts aus
```

Aufgabe:

- Was gibt das Programm im Ausgabebereich aus? Erkläre was passiert.
- Füge dem Vektor ein weiteres Tier hinzu.

Übe Vokabeln

```
val deutsch = Vector("Rechner", "Schildkröte", "Kreis")
val englisch = Vector("computer", "turtle", "circle")
var anzahlRichtig = 0
schleife(5) {
    val i = zufall(deutsch.size)
    val vokabel = deutsch(i)
    val antwort = einlesen("Was heisst " + vokabel + " auf Englisch?")
    if (antwort == englisch(i)) {
        ausgeben("Richtige Antwort!")
        anzahlRichtig = anzahlRichtig + 1
    } else {
        ausgeben("Falsche Antwort. Die richtige Antwort lautet: " + englisch(i))
    }
}
ausgeben("Du hast " + anzahlRichtig + " richtige Antworten gegeben.")
```

Aufgabe:

- Füge mehr Vokabeln ein.
- Übe die Übersetzung vom Englischen ins Deutsche.
- Lasse den Nutzer einstellen, wie viele Vokabeln er üben will. Tipp:
`val anzahl = einlesen("Gebe die Anzahl ein:").toInt`

Hauptstadt-Spiel

```
def hauptstadtSpiel = {
  ausgeben("Willkommen zum Hauptstadt-Spiel!")
  //Eine Map ist eine Abbildung, hier von Landesnamen zu Hauptstadtnamen:
  val stadt = Map("Schweden" -> "Stockholm", "Frankreich" -> "Paris", "Spanien" -> "Madrid")
  var länder = stadt.keySet //keySet liefert die Menge aller Schlüssel in einer Map
  def zufallsLand = scala.util.Random.shuffle(länder.toVector).head
  schleifeSolange(!länder.isEmpty) {
    val land = zufallsLand
    val antwort = einlesen("Was ist die Hauptstadt von " + land + "?")
    ausgeben(s"Dü sagst: $antwort")
    if (antwort == stadt(land)) {
      länder = länder - land //entfernt das Land aus der Menge der Länder
      ausgeben("Richtig! Du hast noch " + länder.size + " Länder!")
    } else ausgeben(s"Falsch! Die Hauptstadt von $land beginnt mit ${stadt(land).take(2)}...")
  }
  ausgeben("DANKE FÜR DIE TEILNAHME! (Drücke ESC)")
}

toggleFullScreenOutput;
setOutputBackground(black); setOutputTextColor(green); setOutputTextFontSize(30)
schleife(100)(ausgeben()) //fülle den Ausgabebereich mit 100 Leerzeilen
hauptstadtSpiel
//AUFGABE: Lege mehr Paare Land -> Stadt an. Messe die Zeit und zähle die Punkte.
```

Erstelle eine Stoppuhr mit **object**

```
object Stoppuhr {  
  def jetzt = System.currentTimeMillis //liefert die aktuelle Zeit in Millisekunden  
  var zeit = jetzt  
  def rüchsetzen = { zeit = jetzt }  
  def gemessen = jetzt - zeit  
  def zufallsWarten(min: Int, max: Int) = //wartet zwischen min und max Sekunden  
    Thread.sleep((zufall(max-min)+min)*1000) //Thread.sleep(1000) wartet 1 Sekunde  
}
```

```
ausgeben("Klicke in den Ausgabebereich und warte...")  
Stoppuhr.zufallsWarten(3, 6) //wartet zwischen 3 und 6 Sekunden  
Stoppuhr.rüchsetzen  
einlesen("Drücke die Eingabetaste, so schnell wie Du kannst.")  
ausgeben("Reaktionszeit: " + (Stoppuhr.gemessen/1000.0) + " Sekunden")
```

Mit **object** kannst Du alles sammeln, was zu einem Objekt gehört.
Du kannst mit einem Punkt auf etwas im Objekt zugreifen: `Stoppuhr.rüchsetzen`

Aufgabe:

- Probiere das Programm aus und messe Deine Reaktionszeit. Wie schnell bist Du?
- Nutze Stoppuhr in der Aufgabe *Zahlenraten* und füge die folgende Ausgabe hinzu:
Richtig geraten! Du hast es mit 5 Versuchen in 32 Sekunden geschafft

Simuliere eine Ampel

```
def alleLöschen = draw(penColor(gray) * fillColor(black) -> PicShape.rect(130,40))
def licht(c: Color, h: Int) = penColor(noColor) * fillColor(c) * trans(20,h) -> PicShape.circle(15)
def leuchtetRot = draw(licht(red, 100))
def leuchtetGelb = draw(licht(yellow, 65))
def leuchtetGrün = draw(licht(green, 30))
def warte(sekunden: Int) = Thread.sleep(sekunden*1000)

leeren; unsichtbar
schleifeSolange (true) { //eine unendliche Schleife
  alleLöschen
  leuchtetRot; warte(3)
  leuchtetGelb; warte(1)
  alleLöschen
  leuchtetGrün; warte(3)
  leuchtetGelb; warte(1)
}
```



Aufgabe:

- Wie wechselt die Ampel? Versuche zu erklären, was hierbei passiert.
- Ändere das Programm so, dass die Ampel doppelt so lang auf grün bleibt.

Steuere die Kröte mit der Tastatur

```
leeren; langsam(0)
activateCanvas()

animate { vor(1) }

onKeyPress {
  case Kc.VK_LEFT => links(5)
  case Kc.VK_RIGHT => rechts(5)
  case Kc.VK_SPACE => vor(5)
  case t =>
    ausgeben("Unbekannte Taste: " + t)
}
```

Aufgabe:

- Schreibe Kc. und drücke Strg+Alt+Leertaste. Dann kannst Du sehen, wie die verschiedenen Tasten heißen.
- Befehle stiftRauf wenn man Pfeil nach oben drückt
- Befehle stiftRunter wenn man Pfeil nach unten drückt
- Befehle stiftfarbe(blau) wenn man B drückt
- Befehle stiftfarbe(rot) wenn man R drückt
- Erhöhe bzw. Verringere die Geschwindigkeit mit den Tasten + bzw. -

Steuere die Kröte mit der Maus

```
leeren; langsam(100)
activateCanvas()
```

```
var zeichnen = true
```

```
onKeyPress {
  case Kc.VK_DOWN =>
    stiftRunter()
    zeichnen = true
  case Kc.VK_UP =>
    stiftRauf()
    zeichnen = false
  case t =>
    ausgeben("Unbekannte Taste: " + t)
}
```

```
onMouseClicked { (x, y) =>
  if (zeichnen) gehen(x, y) else springen(x, y)
}
```

Aufgabe:

- Befehle `füllfarbe(schwarz)` wenn man F drückt
- Führe die Variable `var füllen = true` ein und mache bei Taste `Kc.VK_F`:

```
if (füllen) {
  füllfarbe(schwarz)
  füllen=false
} else {
  füllfarbe(durchsichtig)
  füllen=true
}
```

Erstelle Dein eigenes Bankkonto

```
object meinKonto {  
  val nummer = 123456  
  var stand = 0.0  
  def einzahlen(betrag: Bruchzahl) = {  
    stand = stand + betrag  
  }  
  def abheben(betrag: Bruchzahl) = {  
    stand = stand - betrag  
  }  
  def anzeigen() = {  
    ausgeben("Kontonummer: " + nummer)  
    ausgeben(" Kontostand: " + stand)  
  }  
}
```

```
meinKonto.anzeigen()  
meinKonto.einzahlen(100)  
meinKonto.anzeigen()  
meinKonto.abheben(10)  
meinKonto.anzeigen()
```

Aufgabe:

- Was ist der Kontostand nach Ausführung des Programms? Erkläre was passiert.
- Sorge mit **if** dafür, dass nicht mehr Geld abgehoben werden kann, als auf dem Konto ist.
- Lege eine Konstante **val** maxBetrag = 5000 an und Sorge mit **if** dafür, dass man nicht mehr abheben darf als maxBetrag.

Erstelle viele Objekte mit **class**

Wenn Du viele verschiedene Konten erstellen willst, benötigst Du eine Klasse. Mit **new** kannst Du davon neue Objekte erstellen. Jedes Objekt erhält eine eigene Kontonummer und Kontostand.

```
class Konto(nummer: Ganzzahl) {  
    private var stand = 0.0 //private bedeutet "versteckt"  
    def einzahlen(betrag: Bruchzahl) = {  
        stand = stand + betrag  
    }  
    def abheben(betrag: Bruchzahl) = {  
        stand = stand - betrag  
    }  
    def anzeigen() =  
        ausgeben(s"Konto $nummer: $stand")  
}
```

```
val konto1 = new Konto(12345) //neues Objekt erstellen  
val konto2 = new Konto(67890) //Noch ein Objekt
```

```
konto1.einzahlen(99)  
konto2.einzahlen(88)  
konto1.abheben(57)  
konto1.anzeigen  
konto2.anzeigen
```

Aufgabe:

- Was ist der Stand der beiden Konten, nachdem das Programm ausgeführt wurde? Erkläre was passiert.
- Erstelle weitere Konto-Objekte und zahle Beträge ein und hebe Beträge ab.
- Füge einen Klassenparameter name: Text hinzu, der den Namen des Kontoinhabers aufnehmen soll, wenn Objekte erstellt werden.
- Sorge dafür, dass auch name ausgegeben wird, wenn anzeigen ausgeführt wird.
- Was passiert, wenn Du das befehlst:
konto1.stand = 10000000 ?

Sprich mit Deinem Computer

```
setOutputBackground(black); setOutputTextFontSize(30); setOutputTextColor(green)
ausgeben("Schreibe interessante Antworten, auch wenn die Fragen seltsam sind. Verabschiede Dich mit 'Tschüss'")
def zufällig(xs: Vector[String]) = scala.util.Random.shuffle(xs).head
val aufforderung = Vector("Was bedeutet", "Gefällt Dir", "Wozu brauchen wir", "Erzähl mir mehr über")
var antwort = "?"
val eröffnung = "Worüber möchtest Du reden?"
var worte = Vector("Nabelschnur", "Ketchupeis", "Weihnachtsmann", "Kissenbezüge")
schleifeSolange(antwort != "tschüss") {
  val t = if (antwort == "?") eröffnung
  else if (antwort == "nein") "Nee."
  else if (antwort == "ja") "Nun ja."
  else if (antwort.length < 4) "Oh..."
  else zufällig(aufforderung) + " " + zufällig(worte) + "?"
  antwort = einlesen(t).toLowerCase
  worte = worte ++ antwort.split("[ ,;.:!?]").toList.filter(_.length > 3)
}
ausgeben("Danke für das Gespräch! Jetzt habe ich diese Worte gelernt: " + worte)
```

//Aufgabe:

```
// (1) Probiere das Programm aus und versuche herauszufinden, wie es funktioniert.
// (2) Wann wird die Solange-Schleife beendet?
// (3) Fülle weitere Sätze und Worte in die Vektoren aufforderung und worte.
// (4) Erweitere die Reaktion auf kurze Antworten über "nein" und "ja" hinaus.
```

Verändere das Tischtennis-Spiel

Aufgabe:

- Wähle im Menü Beispiele > Animationen und Spiele > Tischtennis. Probiere es aus!
- Du kannst es steuern mit: Pfeil nach oben/unten und A/Z.
- Drücke ESC um das Spiel zu beenden und untersuche das Programm. Es nutzt die englischen Kojo-Befehle.
- Ändere das Programm so, dass man mit Y statt Z den linken Schläger nach unten bewegen kann. Dann ist das Spiel bei einer deutschen Tastatur bequemer für den linken Spieler.
- Ändere das Programm so, dass der Ball größer wird.
- Ändere das Spielfeld zu einem Tennistisch mit grünem Hintergrund, weißen Linien und einem gelben Ball.

