Kurzanleitung Jython

Inhalt**:**

Jarka Arnold

März 2013

1. Was ist Python/Jython

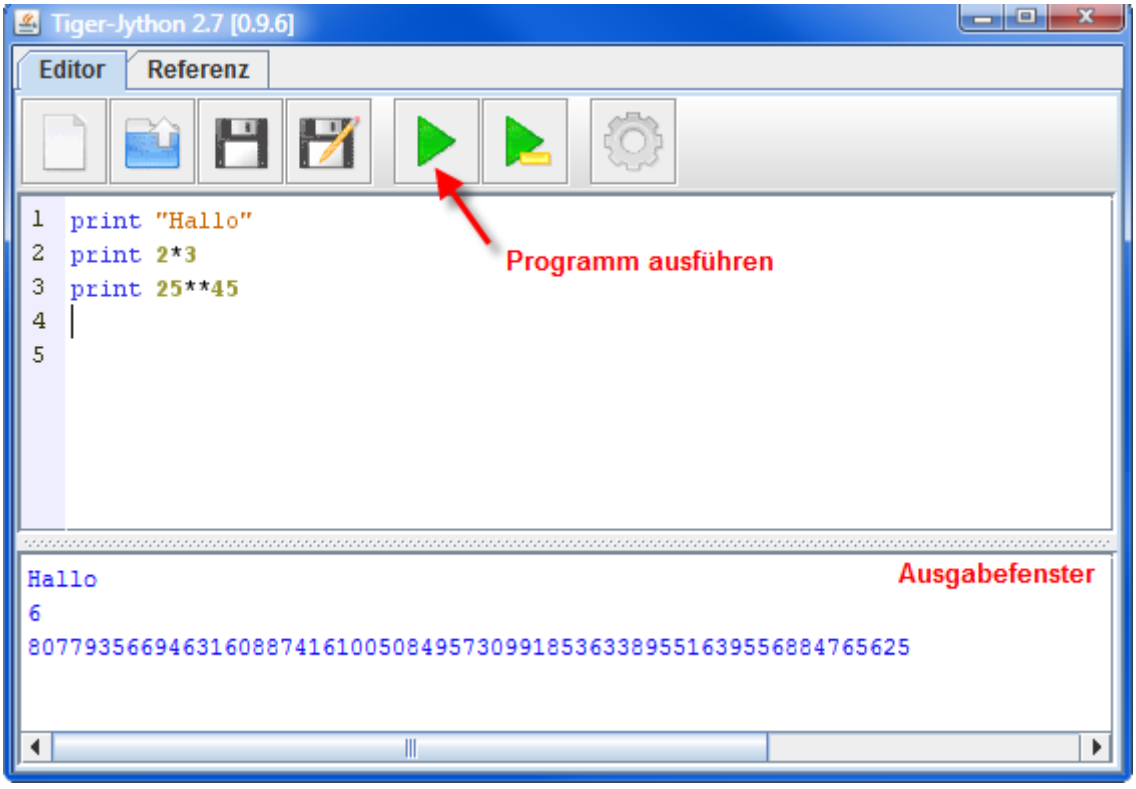
Python ist eine plattformunabhängige interpretierte Programmiersprache, die leistungsstark und einfach zu erlernen ist. Python verfügt über eine klare und übersichtliche Syntax, benötigt keine explizite Typendeklarationen, Programmblöcke werden durch gleiche Einrückung und nicht durch Klammern begrenzt. Jython ist eine Python-Implementierung für die Java-Plattform (JVM). Sämtliche Java-Klassenbibliotheken können importiert werden.

Die Jython-Installations-datei kann von http://www.jython.org/downloads heruntergeladen werden. Damit Jython ausgeführt werden kann, muss auf dem Computer JRE (Java Runtime Environment) installiert sein.

Für die Programmiereinsteiger eignet sich hervorragend die **Entwicklungsumgebung TigerJython** von Tobias Kohn. Diese beinhaltet Jython, einen einfachen Editor, sowie mehrere Klassenbibliotheken (Turtlegrafik, Koordinatengafik, Spielprogrammierung usw.) von Aegidius Plüss (www.aplu.ch). Die Distribution vom TigerJython umfasst eine einzige JAR-Datei, die kostenlos von der Webseite http://www.tobiaskohn.ch/jython heruntergeladen werden kann. Speichern Sie diese JAR-Datei in einem beliebigen Verzeichnis auf der Festplatte und erstellen Sie einen Link auf diese Datei. Im gleichen Verzeichnis können Sie auch alle unsere Musterbeispiele speichern (JythonExamples.zip). Zum Starten von TigerJython genügt es, auf den Link zu klicken.

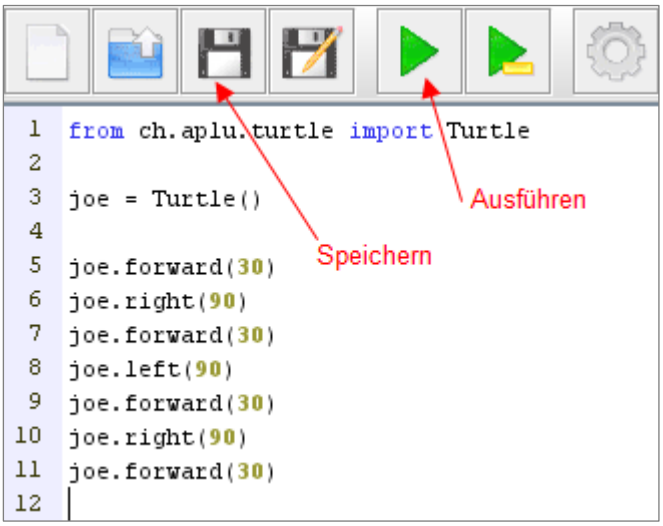
2. Jython Entwicklungsumgebung

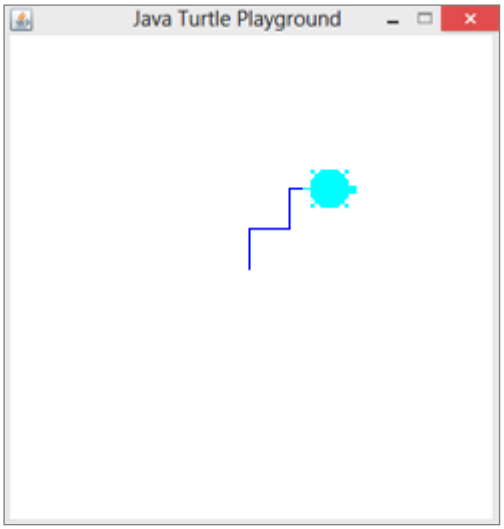
Starten Sie den TigerJython-Editor mit Doppelklick auf *tygerjyton.jar*. Die Bedienung des Editors ist einfach. Es stehen Ihnen die Schaltflächen *Neues Programm, Programm öffnen, Programm speichern, Speichen unter, Programm ausführen, Selektierten Programmteil ausführen und Einstellungen* zur Verfügung. Testen Sie die Funktionalität, in dem Sie einige print-Befehle eingeben und auf die Schaltfläche *Programm ausführen* klicken. Im Unterschied zu vielen andere Programmiersprachen rechnet Jython mit beliebig vielen Dezimalstellen.



3. Turtlegrafik mit Jython

Mit Turtlegrafik lassen sich einfach und anschaulich die wichtigsten Programmiertechniken aufzeigen. Mit dem Import der **Klassenbibliothek Turtle** stehen Ihnen alle Methoden der Turtlegrafik zur Verfügung. Eine Methodenübersicht finden Sie im Anhang. Geben Sie im Editor folgende Zeilen ein und klicken Sie auf ausführen.
Beim Editieren können Sie die üblichen Shortcuts verwenden: Ctrl+C (kopieren), Ctrl+V (einfügen); Ctrl+X (ausschneiden), Ctrl+A (alles markieren), Ctrl+Z (rückgängig).





4. Iteration (while- und for-Schleife)

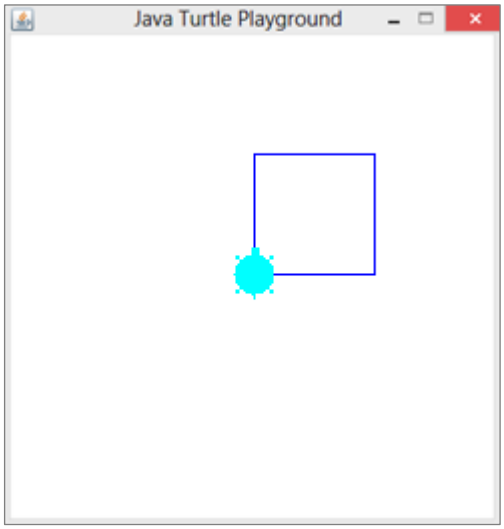
while-Schleife

Tu2.py

```
from ch.aplu.turtle import Turtle

joe = Turtle()

i = 0
while i < 4:
   joe.forward(100)
   joe.right(90)
   i = i + 1
```



Beachten Sie:

- Die **Variablen** müssen nicht deklariert werden
- Die Gross- Kleinschreibung wird beachtet
- Nach dem Schüsselwort *while*, folgten eine Schleifenbedingung und ein **Doppelpunkt**
- Die Zeilen im Rumpf der Schleife müssen **eingerückt** sein
- Kommentarzeilen werden mit # eingeleitet.

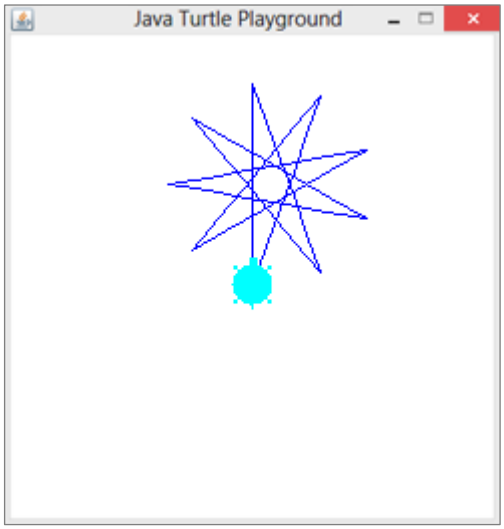
for-Schleife

Tu3.py

```
from ch.aplu.turtle import Turtle

joe = Turtle()

for i in range(9):
   joe.forward(160)
   joe.right(160)
```



Allgemeine Form:
for in in **range**(Startwert, Bedingung, Wertänderung).
Ist der Startwert 0 und die Wertänderung 1, können diese Parameter weggelassen werden.

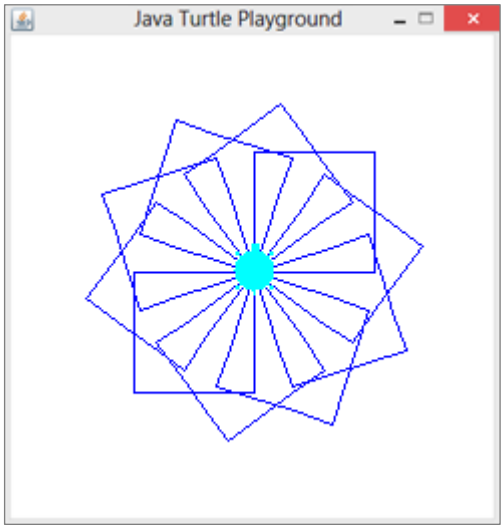
Mehrfache for-Schleifen

Tu3a.py

```
from ch.aplu.turtle import Turtle

joe = Turtle()

for k in range(10):
   for i in range(4):
      joe.forward(80)
      joe.right(90)
   joe.left(36)
```

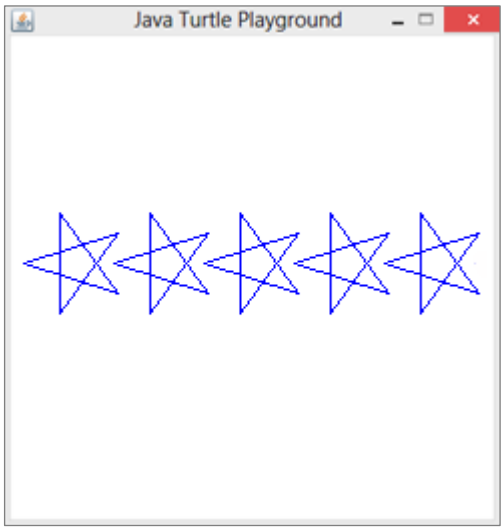


Tu3b.py

```
from ch.aplu.turtle import Turtle

t = Turtle()

for x in range(-160, 160, 70):
   t.setPos(x, 0)
   for i in range(5):
      t.forward(80)
      t.right(144)
```



Die x und y Koordinaten haben Werte von -200 bis 200. Mit der Methode setPos(x, y) wird die Turtle positioniert. Range-Parameter:

```
for x in range(-160, 160, 70)
```

Startwert
x = -160

Schleifenbedingung
x < 160

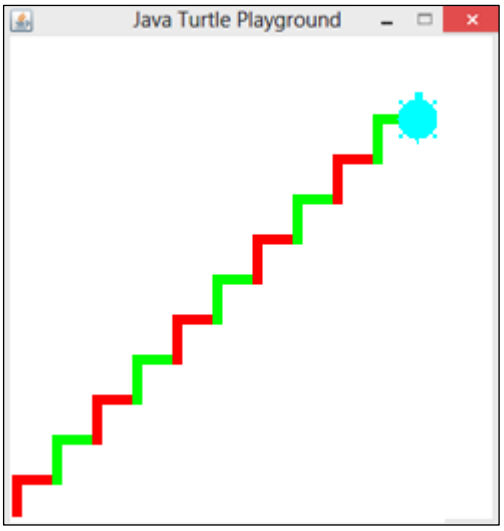
Wertänderung
x = x + 70

5. Selektion (if – else – Struktur)

Tu5.py

```
from ch.aplu.turtle import Turtle

joe = Turtle()
joe.setPos(-200, -200)
joe.setLineWidth(5)
for i in range(10):
  if i % 2 == 0:
     joe.setPenColor("red")
  else:
     joe.setPenColor("green")
  joe.fd(40).rt(90).fd(40).lt(90)
```



Falls die Bedingung nach *if* erfüllt ist, wird der erste Anweisung-Block ausgeführt, sonst die Anweisungen nach *else*. In der letzten Zeile werden die Methoden *forward(), right()*… in der Kurzschreibweise *fd(), rt()* verwendet.

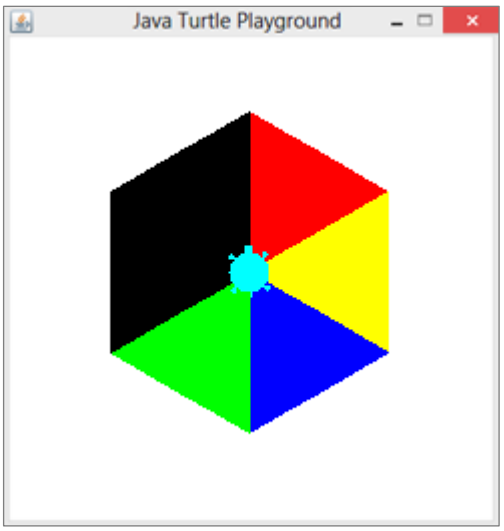
Mehrfache Selektion

Tu6.py

```
from ch.aplu.turtle import Turtle

joe = Turtle()
joe.fillToPoint(0, 0)
for k in range(6):
   if k == 0:
      joe.setPenColor("red")
   elif k == 1:
      joe.setPenColor("yellow")
   elif k == 2:
      joe.setPenColor("blue")
   elif k == 3:
      joe.setPenColor("green")
   else:
      joe.setPenColor("black")

   for i in range(3):
      joe.forward(100)
      joe.right(120)
   joe.right(60)
```



elif ist eine Abkürzung von *else if*
Falls die Bedingung nach if nicht erfüllt ist, wird die Bedingung nach elseif überprüft

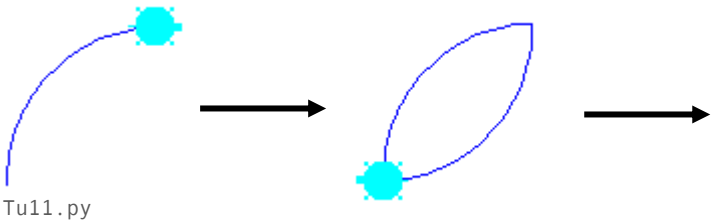
Bei der Verwendung von **logischen Operatoren** sind keine Klammern notwendig:

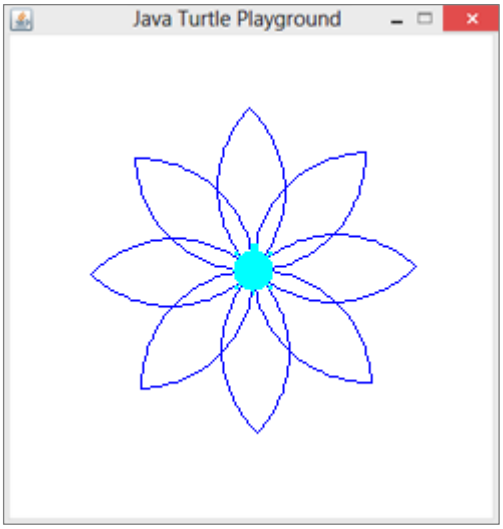
```
Bsp:   if x > 0 and y > 0:
       if x > 5 or x < -5:
```

6. Funktionen

Eine Funktion wird mit dem Schüsselwort ***def*** eingeleitet und mit dem Funktionsnamen aufgerufen und ausgeführt.

Im folgenden Beispiel wird eine Funktion *arc()* definiert, die einen Kreisbogen zeichnet und eine Funktion *petal(),* die eine Blumenblatt zeichnet





Tu11.py

```
from ch.aplu.turtle import Turtle

def arc():
   for i in range(30):
      t.forward(4)
      t.right(3)

def petal():
   arc()
   t.right(90)
   arc()

t = Turtle()
for k in range(8):
   petal()
   t.right(45)
```

Funktion *arc()*

Funktion *petal()*

Aufruf der Funktion *petal()*

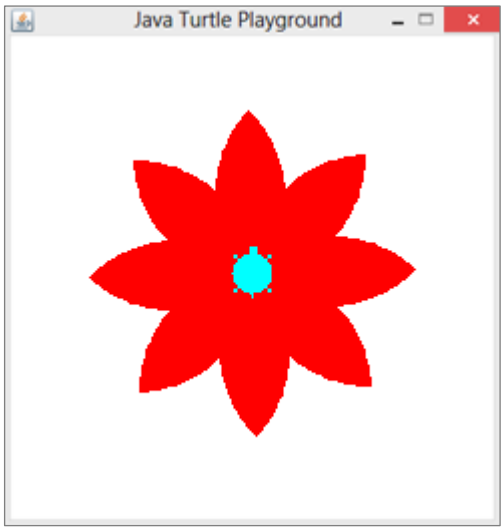
Tu11a.py

```
from ch.aplu.turtle import Turtle

def arc():
   for i in range(30):
      t.forward(4)
      t.right(3)

def petal():
   arc()
   t.right(90)
   arc()

t = Turtle()
t.setPenColor("red")
t.fillToPoint(0, 0)
for k in range(8):
   petal()
   t.right(45)
```



Figur mit einer roten Farbe färben

Funktionen mit Parametern

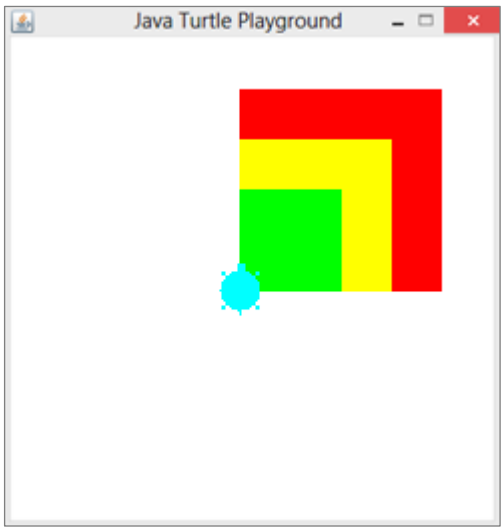
Funktionen können Parameter (auch als Argumente bezeichnet) enthalten, die es ermöglichen, Werte an Funktionen zu übergeben. Parameter benötigen keine Typendeklarationen.

Tu12b.py

```
from ch.aplu.turtle import Turtle

def square(size, color):
   t.setPenColor(color)
   for i in range(4):
      t.forward(size)
      t.right(90)

t = Turtle()
t.fillToPoint(0, 0)
square(120, "red")
square(90, "yellow")
square(60, "green")
```



7. Zufallszahlen

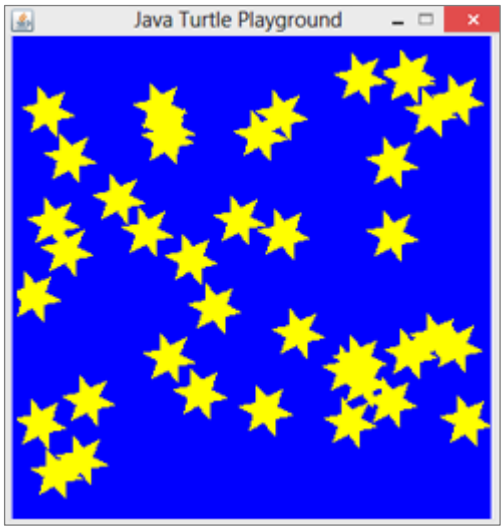
Um Zufallszahlen verwenden zu können, muss die Python-Klasse *random* importiert werden. Die Methode ***random.random()*** liefert eine dezimale Zufallszahl zwischen 0 und 1. Ganzzahlige Zufallszahlen können mit der Methode ***random.randint(a, b)***, wobei a die kleinste und b die grösste Zufallszahl ist. *random.randint(1, 6)* liefert beispielsweise die Würfelzahlen. Im folgenden Beispiel werden mit Hilfe von Zufallszahlen 50 Sterne an zufälligen Positionen gezeichnet.

Tu13.py

```
from ch.aplu.turtle import Turtle
import random

def star(x, y):
   t.setPos(x, y)
   t.fillToPoint(x, y);
   for i in range(6):
      t.forward(10)
      t.right(140)
      t.forward(10)
      t.left(80)

t = Turtle()
t.hideTurtle()
t.clear("blue")
t.setPenColor("yellow")
for i in range(50):
    x = -180 + 360 * random.random();
    y = -180 + 360 * random.random();
    star(x, y)
```



Die Methode clear("blue") übermalt den Hintergrund mir der blauen Farbe.

8. Rekursionen

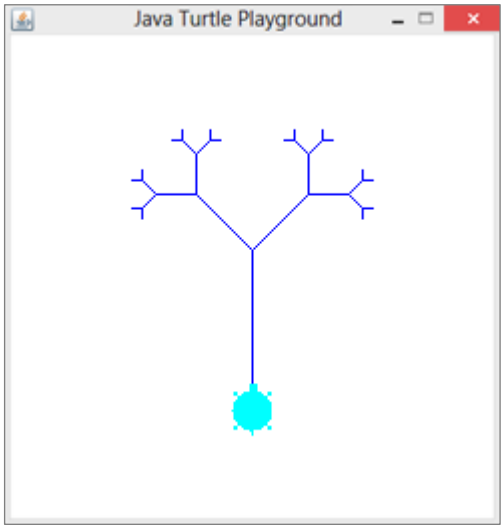
Rekursion ist ein Lösungsverfahren, bei dem ein Problem auf das gleichartige, aber etwas vereinfachte Problem zurückgeführt wird. In Jython ist eine Funktion dann rekursiv, wenn in ihrem Deklarationsteil sie selber aufgerufen wird. Damit sich ein rekursives Programm nicht endlos aufruft, braucht es eine Abbruchbedingung (Verankerung).

Tree.py

```
from ch.aplu.turtle import Turtle

def tree(s):
    if s < 8:
      return
    t.forward(s)
    t.left(45)
    tree(s/2)
    t.right(90)
    tree(s/2)
    t.left(45)
    t.back(s)

t = Turtle()
t.setY(-100)
tree(128)
```

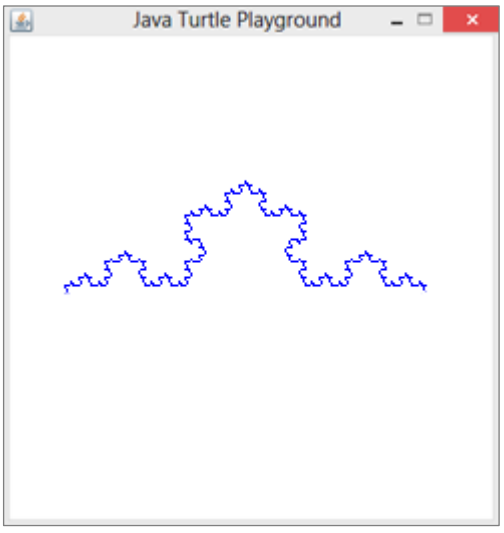


Koch.py

```
from ch.aplu.turtle import Turtle

def koch(s, n):
    if n == 0:
       t.forward(s)
       return
    koch(s / 3, n - 1)
    t.left(45)
    koch(s / 3, n - 1)
    t.right(90)
    koch(s / 3, n - 1)
    t.left(45)
    koch(s / 3, n - 1)

t = Turtle()
length = 200
generations = 4
t.hideTurtle()
t.setPos( -180, 0)
t.right(90)
koch(length, generations)
```



Weitere Beispiele zu Rekursionen mit Turtlegrafik finden Sie auf unserer Website www.turtlegrafik.ch.

9. Tastatur- und Mausevents

Bei Applikationen mit einer grafischen Benutzeroberfläche wird der Programmablauf in der Regel durch Ereignisse (Events) gesteuert. Jede Benutzeraktion mit der Maus oder mit der Tastatur löst einen Event aus.

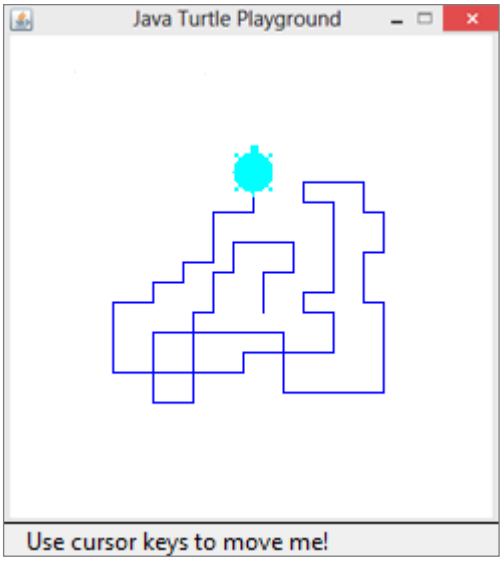
Ein Tastendruck wird mit der Methode *keyPressed()* registriert. Mit der Methode *getKeyCode()* kann zusätzlich bestimmt werden, welche Taste gedrückt wurde.

Tu24.py

```
from ch.aplu.turtle import Turtle
from java.awt.event import KeyEvent

def keyPressed(evt):
   keyCode = evt.getKeyCode();
   if keyCode == KeyEvent.VK_LEFT:
      t.setHeading(-90).fd(10)
   elif keyCode == KeyEvent.VK_RIGHT:
      t.setHeading(90).fd(10)
   elif keyCode == KeyEvent.VK_UP:
      t.setHeading(0).fd(10)
   elif keyCode == KeyEvent.VK_DOWN:
      t.setHeading(180).fd(10)

t = Turtle(keyPressed = keyPressed)
t.speed(-1)
t.addStatusBar(20)
t.setStatusText("Use cursor keys!")
```



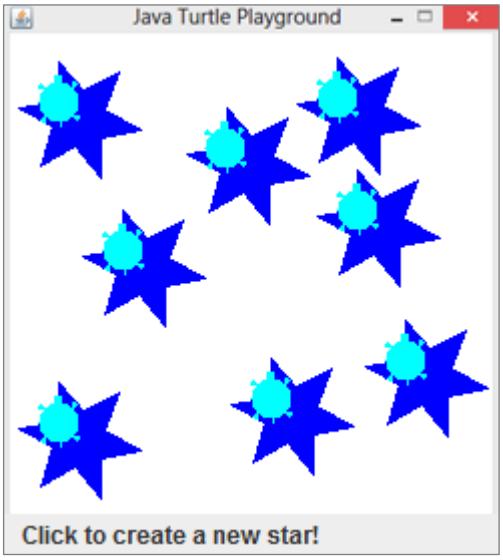
Im nächsten Beispiel wird mit der Methode *mousePressed()* an der Position des Mausklicks eine neue Turtle erzeugt. Weitere Beispiel zu Mausevents sowie ausführlichere Erklärungen zum Programmcode finden Sie auf unserer Turtlegrafik-Website unter Events.

Tu25.py

```
from ch.aplu.turtle import Turtle
from ch.aplu.turtle import TurtleFrame
from thread import start_new_thread

def mousePressed(e):
   t = Turtle(tf)
   x = t.toTurtleX(e.getX())
   y = t.toTurtleY(e.getY())
   t.setPos(x, y)
   start_new_thread(draw, (t, x, y))

def draw(t, x, y):
   t.fillToPoint(x, y);
   for i in range(6):
      t.fd(20).rt(140).fd(20).lt(80)
tf = TurtleFrame(mousePressed = mousePressed)
tf.addStatusBar(20)
tf.setStatusText("Click to create a star")
```



10. Dateneingabe und einfache Berechnungen

Der Wert einer Variablen kann im Programmcode festgelegt werden. Oft wird aber bevorzugt, den Wert interaktiv einzulesen. Mit der Python-Funktion ***raw_input()*** wird eine Zeichenkette eingelesen. Für Zahlen ist eine Umwandlung in den entsprechenden Datentyp nötig.

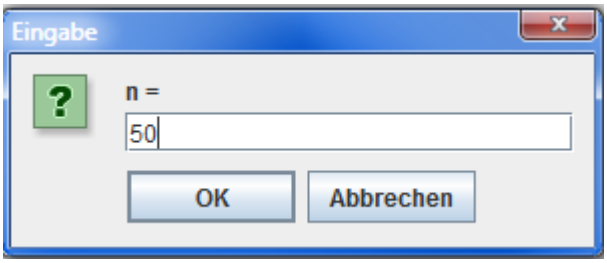
```
x = int(raw_input())          Eingabe von ganzen Zahlen
x = float(raw_input())        Eingabe von Dezimalzahlen
```

Bsp 1: Berechnet die Summe aller ungeraden Zahlen kleiner n

Summe.py

```
print "Summe aller ungeraden Zahlen kleiner als n."
s = 0
i = 1
n = int(raw_input("n = "))
while i < n:
    s = s + n
    i = i + 2
print "Die Summe ist: ", s
```

Nach dem Programmstart erscheint eine Inputbox, in der die Zahl n eingegeben werden kann.
Mit Klick auf OK wird die Summe berechnet.
Das Ergebnis erscheint im Ausgabefenster.




```
Summe aller ungeraden Zahlen kleiner als n.
Die Summe ist:  1250
```

Bsp 2: Ggt Berechnung

Ggt.py

```
print "Berechnet Ggt"
r = 1
a = int(raw_input("1. Zahl = "))
b = int(raw_input("2. Zahl = "))
while r != 0:
    r = a % b
    a = b
    b = r
print "Ggt ist: ", a
```

Bsp. 3: Berechnung der Zahl PI mit der Monte-Carlo Methode

Auf ein Quadrat mit der Seitenlänge 1 werden n Zufallstropfen gestreut. Aus dem Verhältnis der Zahl der Tropfen die innerhalb des Kreises liegen und Anzahl Tropfen im ganzen Quadrat lässt sich die Zahl Pi berechnen:

$$\frac{Fläche\ des\ Viertelkreises}{Fläche\ des\ Quadrats} = \frac{Anzahl\ Treffer}{Anzahl\ aller\ Punkte} \qquad PI = \frac{4 * Anzahl\ Treffer}{Anzahl\ aller\ Punkte}$$

Pi.py

```
import random

print "Berechnet die Zahl Pi"
hits = 0
i = 0
n = int(raw_input("Anzahl Zufallspunkte = "))
while i < n:
   x = random.random()
   y = random.random()
   if x*x + y*y < 1:
     hits = hits + 1
   i = i + 1
pi = 4 * hits / n
print "Pi ist: ", pi
```

Ergebnis für n = 50 000:

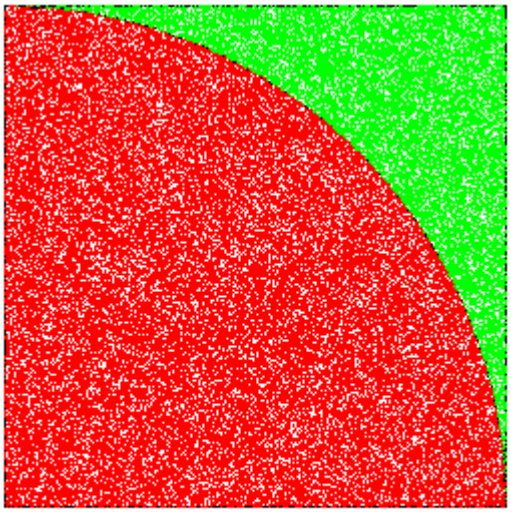
```
Berechnet die Zahl Pi
Pi ist:  3.14228
```

Pi2.py

```
from ch.aplu.util import GPanel
from java.awt import Color
import random

p = GPanel(-0.5, 1.5, -0.5, 1.5)
p.clear();
p.rectangle(0, 0, 1, 1);
p.arc(1, 0, 90);
hits = 0
i = 0
n = int(raw_input("Anzahl Punkte = "))
while i < n:
   x = random.random()
   y = random.random()
   if x*x + y*y < 1:
     hits = hits + 1
     p.color(Color.red)
   else:
     p.color(Color.green)
   i = i + 1
   p.point(x, y)
pi = 4 * hits / n
p.color(Color.black)
p.pos( 0, -0.2 )
p.text("Die Zahl PI ist  " + str(pi))
```

Um das Problem auch grafisch lösen zu können, kann man die Klassenbibliothek *GPanel* importieren. Mehr zur Koordinatengrafik finden Sie auf unserer Website www.java-online.ch/gpanel



11. Anhang 1: Turtle-Grafik Methodenübersicht

Methode	Aktion
forward(distanz)	bewegt Turtle vorwärts
back(distanz)	bewegt Turtle rückwärts
left(winkel)	dreht Turtle nach links
right(winkel)	dreht Turtle nach rechts
hideTurtle()	versteckt Turtle (Turtle zeichnet schneller)
showTurtle()	zeigt Turtle
home()	setzt Turtle in die Mitte , Richtung nach oben
speed(zahl)	setzt Turtlegeschwindigkeit
penUp()	hebt Zeichenstift auf (Spur unsichtbar)
penDown()	setzt Zeichenstift ab (Spur sichtbar)
setColor(color)	legt Turtlefarbe fest
setPenColor(color)	legt Stiftfarbe fest
setFillColor(color)	legt Füllfarbe fest
fill()	füllt eine vorhandene, geschlossene Figur, in der sich die Turtle befindet
fill(x , y)	füllt eine geschlossene Figur, in der sich der Punkt (x, y) befindet
fillToPoint(x , y)	füllt fortlaufend die gezeichnete Figur vom Punkt (x, y) aus
fillToHorizontal(y)	füllt die Fläche zwischen der Figur und der gegebenen horizontalen Linie
fillToVertical(x)	füllt die Fläche zwischen der Figur und der gegebenen vertikalen Linie
setPos(x, y)	setzt Turtle auf die Position (x, y)
getPos()	gibt die Turtleposition zurück
distance(x, y)	gibt die Entfernung von (x, y) zurück
clean()	löscht die Zeichnung
clear(color)	löscht die Zeichnung und übermalt das ganze Fenster mit der gegebenen Farbe
label(text)	schreibt Text an der aktuellen Position
setFont(Font font)	legt Schriftart fest
setFontSize(size)	legt Schriftgrösse fest
addStatusBar(20)	fügt eine Statusbar mit der Höhe 20 Pixel unten am Turtlefenster hinzu
setStatusText("Press any key!")	schreibt eine Mitteilung in die Statusbar

Die ganze Dokumentation zu der Java-Klassenbibliothek Turtle sehen Sie unter
http://www.aplu.ch/classdoc/turtle/index.html

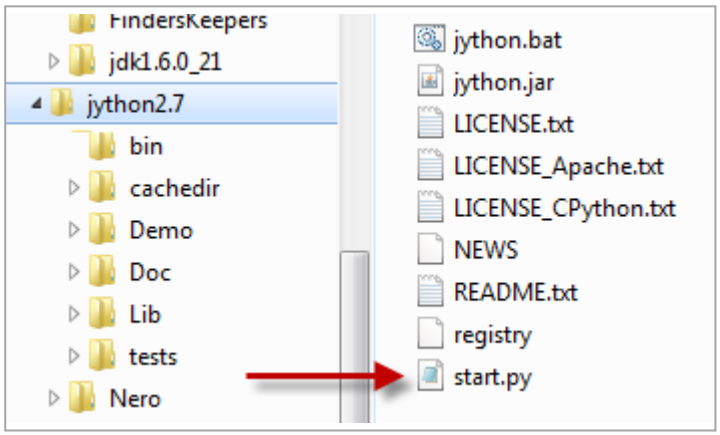
12. Anhang 2: Jython-Installation mit Original-Distributionen

Falls Sie die von uns empfohlene **Programmierumgebung *TigerJython*** verwenden, ist diese Installation **nicht notwendig** (siehe Seite 1). Im Folgenden wird die Installation mir der Original-Jython-Distribution beschrieben:

1. Laden Sie die Jython-Installationsdatei von der Webseite www.jython.org/downloads herunter. Wir empfehlen die Version 2.7 zu verwenden.
2. Damit die jar-Datei ausgeführt werden kann, muss auf dem Computer JRE (Java Runtime Environment) installiert sein (www.oracle.com/technetwork/java/javase/downloads).
3. Installieren Sie Jython mit Doppelklick auf die Installationsdatei z. Bsp. im Verzeichnis c:\Programms\Jython2.7
4. Laden Sie folgende Java-Klassenbibliotheken herunter und speichern Sie diese z. Bsp. unter c:\classes

aplu5.jar	http://www.aplu.ch/download
NxtJLib.jar	http://www.aplu.ch/nxt
bluecove-2.1.0.jar	http://www.bluecove.org
JGameGrid.jar	http://www.aplu.ch/jgamegrid

5. Erstellen Sie im Jython-Verzeichnis eine neue Textdatei **start.py** mit untenstehenden Einträgen. Diese Datei bewirkt den automatischen Import der Java-Klassenbibliotheken.




```
from __future__ import division
import sys
sys.path.append("c:/classes/aplu5.jar")
sys.path.append("c:/classes/bluecove-2.1.0.jar")
sys.path.append("c:/classes/JGameGrid.jar")
sys.path.append("c:/classes/NxtJLib.jar")
from ch.aplu.util import *
from ch.aplu.turtle import *
from ch.aplu.jgamegrid import *
from math import *
import random
from java.awt import Color
print "Loaded: math,division,random,GPanel,Turtle,NxtJLib,JGameGrid,Color"
```

6. Die Startdatei **jy.bat** erstellen.
   Damit Sie nicht bei jedem Jython-Start in das Jython-Verzeichnis wechseln müssen, ist es praktisch, eine Startdatei mit folgendem Eintrag zu erstellen:

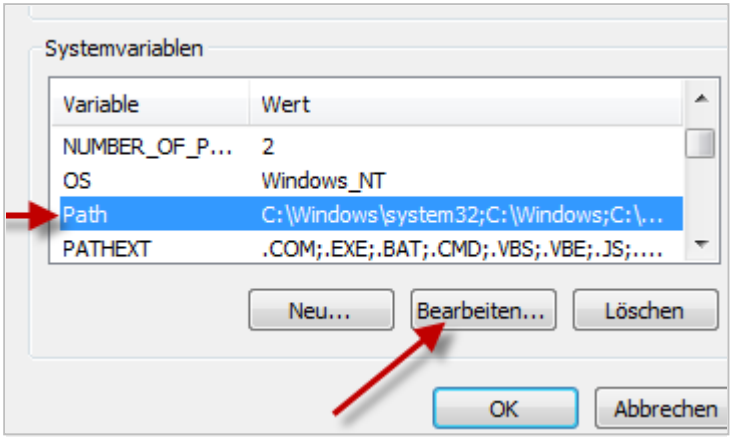
```
java -jar c:\programs\jython2.7\jython.jar -i c:\programs\jython2.7\start.py
```

   Anmerkung: Jython verwendet ein Verzeichnis c:\programs\jython2.5.1\cachedir\packages, in welchen temporäre Dateien gespeichert werden. Auf dieses Verzeichnis ist daher Schreibrecht erforderlich. Bei einem eingeschränkten Benutzerkonto (keine Administratorrechte) kann dieses Verzeichnis im Home-Verzeichnis des Benutzers angelegt werden. Dazu ist jy.bat wie folgt zu ergänzen (alles auf einer Zeile):

   java -Dpython.cachedir="%HOMEDRIVE%%HOMEPATH%" -jar c:\programs\jython2.7\jython.jar -i c:\programs\jython2.7\start.py

   Damit die Startdatei von überall ausgeführt werden kann, kopiert man diese in ein Verzeichnis, welches im Pfad liegt. Sie können ein neues Verzeichnis erstellen, z. Bsp. *c:\bat.* Wählen Sie Systemsteuerung /System/Erweiterte Systemeinstellungen/Umgebungsvariablen

   Ergänzen Sie den Path mit dem Eintrag C:\bat;



6. Testen Sie die Installation:
   Wählen Sie die Eingabeaufforderung aus dem Windows-Zubehör. Es empfiehlt sich, diese in der Taskleiste hinzuzufügen 

   Geben Sie *jy* ein. Jython wird gestartet
   Testen Sie den Import von Java-Klassenbibliotheken z. Bsp. mit einem Turtle-Programm

```
>>>t = Turtle()
>>>t.forward(100)
```

7. Programme speichern und editieren

   Jython-Programme können auch mit einem Texteditor editiert und anschliessend mit dem Befehl ***excefile(filename)*** ausgeführt werden. Als Editor kann z. Bsp. **Notepad** oder **Notepad++** verwendet werden. Im Notepad++ werden die Schlüsselwörter farbig markiert. Dadurch wird der Programmcode übersichtlicher. Die Installationsdatei kann gratis vom Internet
   (http://notepad-plus-plus.org/) heruntergeladen werden.

Bsp. 1:
Erstellen Sie auf Ihrer Festplatte ein neues Verzeichnis z. Bsp: d:\jy. Öffnen Sie den Texteditor und geben Sie folgendes Programm ein:

```
i = 0
while i < 10:
  print i
  i = i + 1
```

Speichern Sie diese Datei unter d:/jy/ex1.py

Wechseln Sie zur *Eingabeaufforderung*, in dem Jython gestartet ist und geben Sie ein:
>>> execfile("d:/jy/ex1.py")
Das Programm kann durch wiederholte Eingabe von execfile("d:/jy/ex2.py") oder einfacher durch die Betätigung der Pfeil-Taste [↑] mehrmals ausgeführt werden.

Bsp. 2: Erstellen Sie im Texteditor eine zweite Datei d:/jy/ex2.py

```
x = random.randint(1, 10)
if x < 3:
  print "Sie gewinnen", 2 * x, "Franken"
elif x < 7:
  print "Sie gewinnen", 3 * x, "Franken"
else:
  print "Sie gewinnen", 5 * x, "Franken"
```

Führen Sie das Programm mit >>> execfile("d:/jy/ex2.py") aus.

Bsp. 3
Geben Sie im Texteditor das Programm zur Berechnung des Ggt's ein und speichern Sie es z. Bsp. unter d:/jy/ex3.py.

```
def ggt(a, b):
  r = 1
  while r != 0:
    r = a % b
    a = b
    b = r
  print "Ggt ist: ", a
```

Führen Sie nun das Programm mit
execfile("d:/jy/ex3.py")
aus. Jython kennt jetzt die Funktion ggt(). Diese kann

```
>>> execfile("d:\jy\ex3.py")
>>> ggt(35, 14)
Ggt ist:  7
>>> ggt(154, 126)
Ggt ist:  14
```