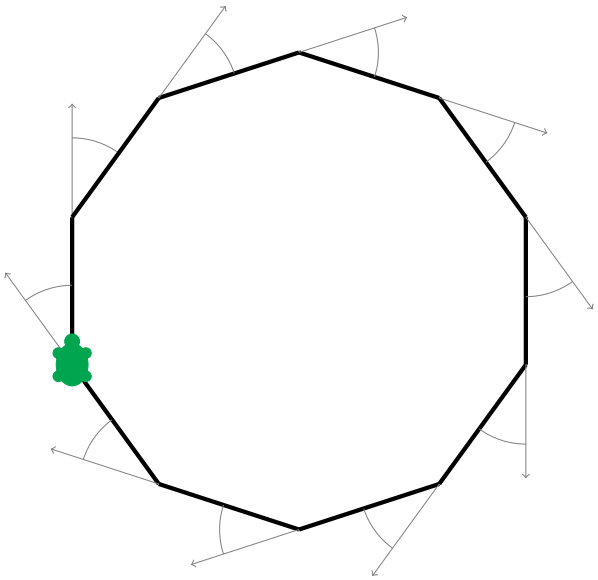
ABZ AUSBILDUNGS- UND BERATUNGSZENTRUM FÜR INFORMATIKUNTERRICHT

ETH zürich

Urs Hauser Juraj Hromkovič Tobias Kohn Dennis Komm Giovanni Serafini

Programmieren mit Python



Schweizerische Eidgenossenschaft
Departement für Wirtschaft, Bildung und Forschung

Departementstagung, 15. Februar 2018

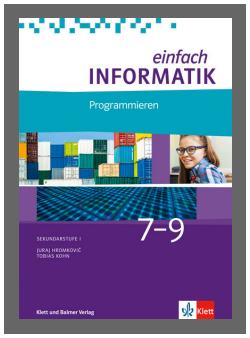
Programmierumgebung

Die vorliegenden Unterrichtsunterlagen wurden für die Programmierumgebung TigerJython (`http://jython.tobiaskohn.ch`) erstellt und sind kostenlos auf `http://www.abz.inf.ethz.ch/maturitatsschulen/unterrichtsmaterialien/` verfügbar.

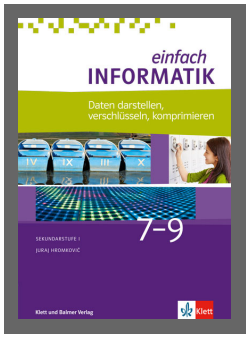
Weiterführende Literatur

Die Lehrwerksreihe „Einfach Informatik“ ermöglicht einen kompetenzorientierten und Lehrplan-21-kompatiblen Unterricht im Bereich Informatik auf den Schulstufen 7–9. Sie besteht aus drei Bänden und ist imKlett-Verlag erhältlich.

`https://www.klett.ch/de/inentwicklung/einfach_informatik/index.php`



Programmieren



Daten darstellen, verschlüsseln, komprimieren

Strategien entwickeln

Nutzungsrechte

Das ABZ stellt diese Lehrmaterialien zur Förderung des Unterrichts interessierten Lehrkräften oder Institutionen zur internen Nutzung kostenlos zur Verfügung.

ABZ

Das Ausbildungs- und Beratungszentrum für Informatikunterricht der ETH Zürich unterstützt Schulen und Lehrkräfte, die ihren Informatikunterricht entsprechend auf- oder ausbauen möchten, mit einem vielfältigen Angebot. Es reicht von individueller Beratung und Unterricht durch ETH-Professoren und das ABZ-Team direkt vor Ort in den Schulen über Ausbildungs- und Weiterbildungskurse für Lehrkräfte bis zu Unterrichtsmaterialien.

`www.abz.inf.ethz.ch`

1 Steuerung der Schildkröte

Ein **Computerbefehl** ist eine Anweisung, die der Computer versteht und ausüben kann. Der Computer kennt eigentlich nur sehr wenige Befehle und alle komplizierten Tätigkeiten, die wir vom Computer vollbracht haben wollen, müssen wir aus den einfachen Computerbefehlen zusammensetzen.

Eine solche Folge von Computerbefehlen nennen wir ein **Programm**.

In diesem Kapitel lassen wir den Computer eine Schildkröte über den Bildschirm bewegen, womit beispielsweise geometrische Figuren gezeichnet werden können.

Aufgabe 1

Tippen Sie das folgende Programm ab und führen Sie es aus, indem Sie auf das Play-Symbol „▶“ drücken.

```
from gturtle import *

makeTurtle()

forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
```

Chapeau! Sie haben die Schildkröte mit Hilfe der Befehle `forward(100)` und `right(90)` ein Quadrat zeichnen lassen. Diese Instruktionen sind in einem für den Computer verständlichen „Wörterbuch“ gespeichert, das wir in jedem Programm zunächst „laden“ müssen:

```
from gturtle import *
```

Anschliessend muss die Schildkröte erstellt werden:

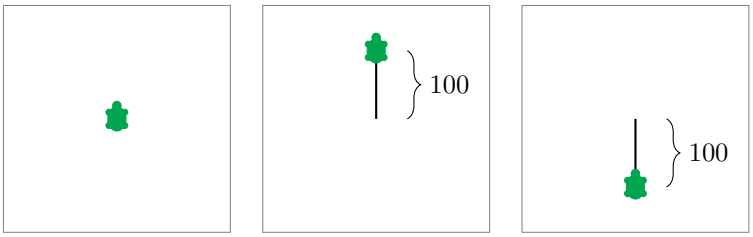
```
makeTurtle()
```

Wichtig

Beachten Sie das „`()`“ am Ende des Befehls; diese Klammern dürfen nicht weggelassen werden. Innerhalb der Klammern werden wir später den Befehlen Werte, z. B. Zahlen, übergeben.

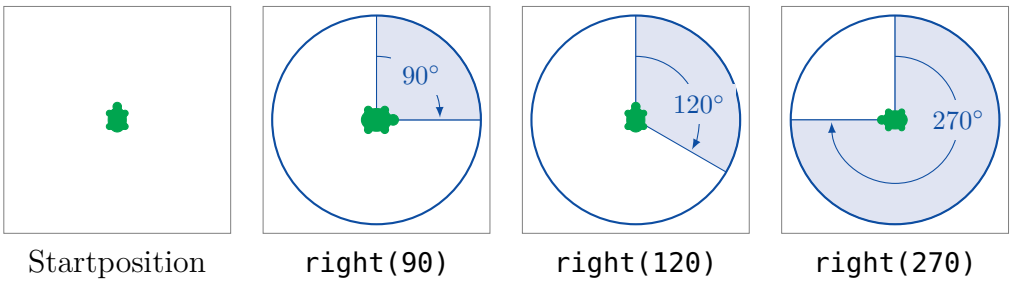
Strecken zeichnen

Mit `forward(100)` und `back(100)` bewegt sich die Schildkröte um `100` Schritte (Pixel auf dem Bildschirm) vorwärts bzw. rückwärts und zeichnet dabei ihre Spur. Die Befehle können mit `fd(100)` und `bk(100)` abgekürzt werden. Sie können die Anzahl der Schritte frei wählen.



Drehen

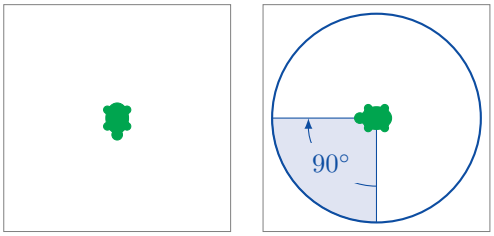
Mit dem Befehl `right(x)` (oder als Abkürzung `rt(x)`) dreht sich die Schildkröte um `x` Grad nach rechts.



Der Befehl `left(x)` bzw. `lt(x)` kann gleichermassen für eine Links-Drehung um `x` Grad verwendet werden.

Wichtig

Das Drehen nach links und rechts bezieht sich immer auf die aktuelle Ausrichtung der Schildkröte, wie das folgende Beispiel mit dem Befehl `right(90)` zeigt.

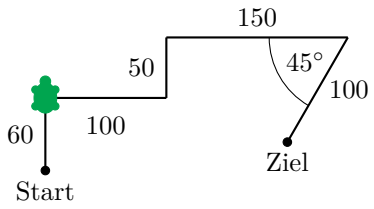


Startposition `right(90)`

Aufgabe 2

Erweitern Sie das folgende Programm so, dass das Bild rechts gezeichnet wird. Die Schildkröte hat hier bereits den ersten Befehl `forward(60)` ausgeführt.

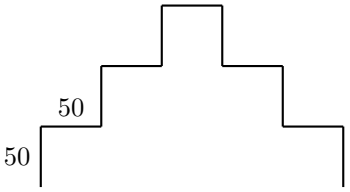
```
from gturtle import *

makeTurtle()

forward(60)
...
```



Aufgabe 3

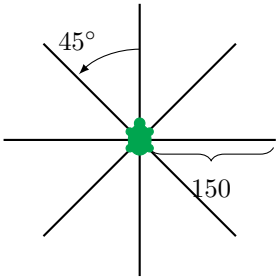
a. Schreiben Sie ein Programm, das das Bild rechts zeichnet.

b. Ändern Sie das Programm so ab, dass es nur die Befehle `forward(50)` und `right(90)` verwendet.



Aufgabe 4

Schreiben Sie ein Programm, das den Stern rechts zeichnet; dies kann im oder gegen den Uhrzeigersinn passieren.



2 Programmablaufstrukturen

Wiederholungen mit `repeat`

Bislang waren die erstellten Programme sehr repetitiv; beispielsweise haben wir in Aufgabe 1 ein Quadrat mit Seitenlänge 100 gezeichnet, indem wir die Befehle `forward(100)` und `right(90)` vier Mal wiederholt haben:

```
from gturtle import *

makeTurtle()
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
```

Der Computer kann die Wiederholung solcher Befehle aber auch direkt übernehmen, ohne dass wir Sie mehrmals eintippen müssen.

Konkret verwenden wir in einem solchen Fall eine sogenannte **`repeat`-Schleife**, mit der wir dem Computer sagen

„Wiederhole folgende Befehle 4 Mal“.

In der Sprache des Computers sieht dies wie folgt aus:

```
from gturtle import *

makeTurtle()

repeat 4:
    forward(100)
    right(90)
```

Direkt hinter **repeat** gibt eine Zahl (hier die 4) an, wie oft die folgenden Befehle ausgeführt werden sollen. Anschliessend muss zwingend ein Doppelpunkt „:“ folgen. Alle Befehle, die wiederholt werden sollen, müssen gleichmässig **eingerückt** werden. Drückt man nach dem Doppelpunkt die Enter-Taste, geschieht dies automatisch. Natürlich kann man hierzu auch die Tabulator-Taste ⇤⇥ verwenden.

Aufgabe 5

Überlegen Sie sich kurz, was das folgende Programm zeichnet. Kürzen Sie es anschliessend mit Hilfe von **repeat** ab.

```
from gturtle import *

makeTurtle()

forward(50)
right(120)
forward(50)
right(120)
forward(50)
right(120)
```

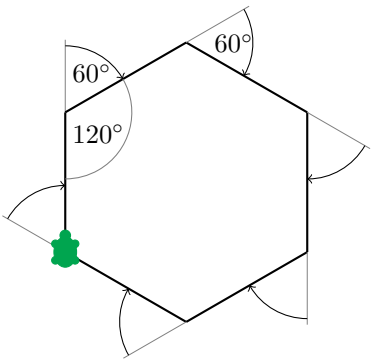
Aufgabe 6

Vereinfachen Sie das Programm aus Aufgabe 4 mit Hilfe einer **repeat**-Schleife.

Ein regelmässiges n-Eck ist ein Vieleck mit n gleich langen Seiten, dessen Innenwinkel alle gleich gross sind.

Aufgabe 7

Schreiben Sie ein Programm, das ein regelmässiges 6-Eck mit Seitenlänge 50 zeichnet. Beachten Sie, dass die Turtle beim Zeichnen des 6-Ecks insgesamt eine 360°-Drehung macht. Verwenden Sie die **repeat**-Schleife.



Aufgabe 8

Schreiben Sie ein Programm, das ein regelmässiges 10-Eck mit Seitenlänge 50 zeichnet. Verwenden Sie dafür die **repeat**-Schleife.

Aufgabe 9

Schreiben Sie nun ein Programm, das ein regelmässiges 360-Eck mit Seitenlänge 2 zeichnet. Welche Beobachtung machen Sie dabei?

Eigene Befehle für die Schildkröte definieren

Wenn wir Programme schreiben wollen, in denen wir viele Quadrate oder andere Figuren brauchen, wird das sehr mühsam und aufwendig. Deshalb bringen wir der Schildkröte mit **def** neue Befehle bei, zum Beispiel `quadrat100()`, der ein Quadrat mit der Seitenlänge 100 zeichnen soll. Diesen versteht die Schildkröte und wir können ihn beliebig oft im Programm aufrufen bzw. verwenden:

```
from gturtle import *

def quadrat100():
    repeat 4:
        forward(100)
        right(90)

makeTurtle()
```

Bei der Ausführung per ▶ beobachten wir allerdings, dass die Schildkröte sich nicht bewegt. Der Grund hierfür ist, dass wir den Befehl `quadrat100()` zwar definiert, aber noch nicht aufgerufen haben. Das machen wir, indem wir ihn unterhalb seiner Definition hinzufügen:

```
from gturtle import *

def quadrat100():
    repeat 4:
        forward(100)
        right(90)

makeTurtle()
quadrat100()
```

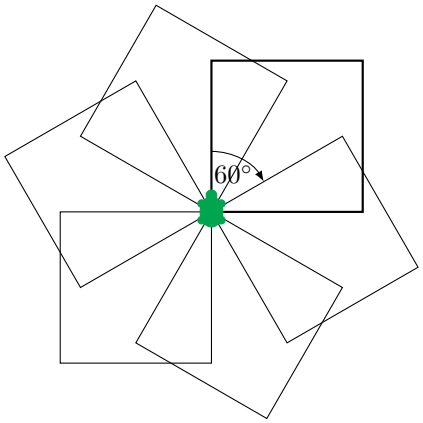
Beim Aufruf des Befehls `quadrat100()` wird der eingerückte Programmteil ausgeführt, also ein Quadrat gezeichnet.

Wichtig

Die Klammern `()` gehören zum Befehl dazu, auch wenn Sie momentan noch leer sind.

Eigene Befehle sind besonders vorteilhaft, wenn wir diese mehrmals verwenden möchten. Zum Beispiel ist dies ersichtlich, wenn wir `quadrat100()` sechsmal aufrufen:

```
from gturtle import *

def quadrat100():
    repeat 4:
        forward(100)
        right(90)

makeTurtle()

repeat 6:
    quadrat100()
    right(60)
```



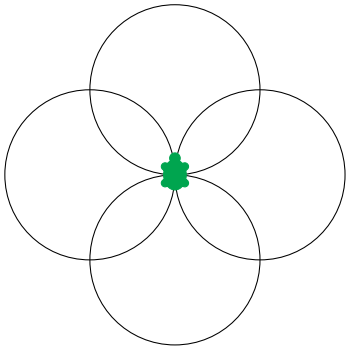
Wir können eine beliebige Anzahl von Befehlen definieren und ausführen lassen.

Aufgabe 10

Definieren Sie einen Befehl `circle()`, der den Kreis aus Aufgabe 9 zeichnet. Rufen Sie diesen Befehl auf.

Aufgabe 11

Benutzen Sie den Befehl `circle()` aus Aufgabe 10, um die abgebildete Figur zu zeichnen, die aus vier Kreisen besteht, die gegenseitig um 90° gedreht sind.



Mit der Definition neuer Befehle erweitern wir letzten Endes das Vokabular der Schildkröte, wobei wir bei der Definition neuer Wörter auf schon bekannte Wörter zurückgreifen.

Nun versteht die Schildkröte beispielsweise die neuen Befehle `quadrat100()` und `circle()`.

Befehle mit Parametern

Sie haben gerade gelernt, wie Sie eigene Befehle definieren und wie Sie ihnen einen Namen Ihrer Wahl geben, um sie dann mit diesem Namen aufzurufen und das gewünschte Bild vom Computer zeichnen zu lassen.

Wenn Sie nun drei Quadrate mit den Seitenlängen 50, 100 und 200 zeichnen möchten, müssen Sie drei Befehle `quadrat50()`, `quadrat100()` und `quadrat200()` schreiben. Dies ist unnötigerweise aufwändig und repetitiv. Die Programmiersprache erleichert uns auch dabei die Arbeit.

Betrachten Sie nun die entsprechenden drei Befehle genauer:

```
def quadrat50():
    repeat 4:
        forward(50)
        right(90)
```

```
def quadrat100():
    repeat 4:
        forward(100)
        right(90)
```

```
def quadrat200():
    repeat 4:
        forward(200)
        right(90)
```

Die drei Befehle sind sich sehr ähnlich und unterscheiden sich nur in den gelben Zahlen `50`, `100` und `200`. Diese Zahlen legen die Anzahl der Schritte fest, die die Schildkröte nach vorne machen soll.

Wir wollen nun einen einzigen Befehl `quadrat()` schreiben, mit dem wir alle möglichen Quadrate zeichnen können:

```
from gturtle import *

def quadrat(schritte):
    repeat 4:
        forward(schritte)
        right(90)

makeTurtle()
quadrat(50)
quadrat(100)
quadrat(200)
```

Was haben wir gemacht?

1. Überall dort, wo die Anzahl der Schritte steht, schreiben wir statt einer konkreten Zahl einen Namen, in diesem Fall `schritte`.
2. Darüber hinaus ergänzen wir die Klammern gleich nach dem Namen des neuen Befehls mit der Angabe `schritte`. Der Computer weiss nun von vornherein, dass wir die Anzahl der Schritte später frei wählen wollen.
3. Beim Aufruf des Befehls muss die Anzahl der Schritte **zwingend** angegeben werden.

Wenn wir, wie im obigen Programm, den Befehl `quadrat(50)` schreiben, setzt der Computer im Befehl überall dort, wo `schritte` steht, die Zahl `50` ein.

```
                50
def quadrat(schritte):
    repeat 4:        50
        forward(schritte)
        right(90)
```

Anschliessend zeichnet der Computer ein Quadrat mit Seitenlänge 50.

Wir nennen `schritte` einen **Parameter**. Im Programm oben sind 50, 100, 200 die jeweiligen **Werte des Parameters** `schritte`. Dabei sagen wir, dass wir dem Befehl `quadrat()` der Reihe nach den Wert 50, den Wert 100 und den Wert 200 **übergeben**. Wir kennen dies bereits vom Befehl `forward(100)`, dem wir den Wert 100 übergeben.

Aufgabe 12

Definieren Sie einen Befehl `sechseck(s)`, der ein regelmässiges Sechseck mit Seitenlänge `s` zeichnet.

Sie können auch Befehle mit mehreren Parametern definieren.

Aufgabe 13

Definieren Sie einen Befehl `vieleck(s,n)`, der ein regelmässiges `n`-Eck mit Seitenlänge `s` zeichnet.

Wir können die neu von uns definierten Befehle wiederum verwenden, um neue Befehle zu definieren:

Aufgabe 14

Was zeichnet das folgende Programm? Tippen Sie es ab und überprüfen Sie ihre Vermutung. *Die Lösung finden Sie in Aufgabe 19 auf Seite 15*

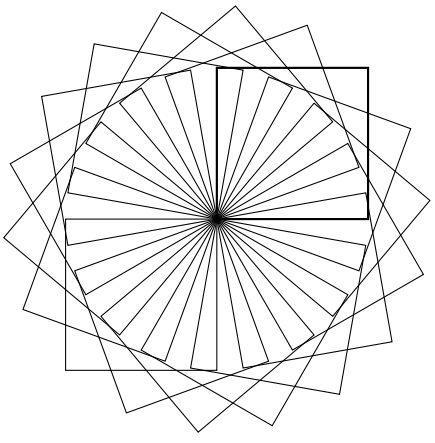
```
from gturtle import *

def figur1(s):
    repeat 3:
        forward(s)
        right(120)

def figur2(s):
    repeat 6:
        figur1(s)
        right(60)

makeTurtle()
figur2(100)
```

Wichtig

Damit das Zeichnen schneller geht, kann man die Schildkröte mit dem Befehl `hideTurtle()` verstecken. Analog kann man eine Schildkröte mit `showTurtle()` wieder anzeigen lassen.

Aufgabe 15

Die abgebildete Sternfigur besteht aus Quadraten, welche um die linke untere Ecke rotiert wurden.



Definieren Sie einen Befehl `rotquads(n)`, der den Befehl `quadrat(50)` verwendet, um die Sternfigur aus `n` Quadraten mit Seitenlänge 50 zu zeichnen.

Benutzen Sie hier auch `hideTurtle()`.

Aufgabe 16

Erweitern Sie den Befehl aus Aufgabe 15 zu `rotquads(n,s)`, wobei `s` die Seitenlänge der gezeichneten Quadrate angibt. Diese Grösse kann direkt via `quad(s)` weitergegeben werden.

3 Ausblick – Animationen und Simulationen

Wir können mit TigerJython Animationen nach dem gleichen Prinzip erstellen, wie wir sie von Zeichentrickfilmen oder Daumenkinos her kennen.

1. Eine Figur **zeichnen**.
2. Die Figur bzw. den Bildschirm nach ein paar Millisekunden **löschen**.
3. Die Figur **erneut zeichnen** – zum Beispiel ganz leicht verschoben oder verdreht.

Wiederholen wir diese drei Schritte sehr oft und schnell, so erscheint das Ergebnis als **Animation**.

Mit dem Befehl `delay(20)` wartet das Programm 20 Millisekunden ($\frac{1}{50}$s) bis es den nächsten Befehl ausführt. Das bedeutet also, dass wir in einer Sekunde ca. 50 Bilder zeichnen.

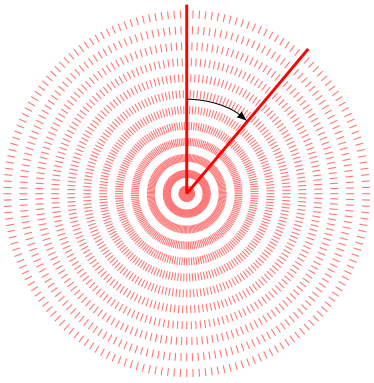
Mit dem Befehl `clear()` löschen wir dann den Bildschirm, wobei die Schildkröte an derselben Position und mit derselben Ausrichtung stehen bleibt.

Mit dem Befehl `setPenColor("red")` setzten Sie die Stiftfarbe auf rot.

Aufgabe 17

Überlegen Sie sich, was die einzelnen Programmteile genau machen und testen Sie das Programm. Was könnte man im Programm abändern, damit der Zeiger eine ganze Kreisrotation in *ungefähr* einer Sekunde schafft?

```
from gturtle import *

makeTurtle()
hideTurtle()
setPenColor("red")

repeat 1000:
    forward(100)
    back(100)
    delay(20)
    clear()
    right(3)
```



Es kann sein, dass die Animationen etwas flackern oder ruckeln. Dies liesse sich beseitigen, würde hier aber den zeitlichen Rahmen sprengen.

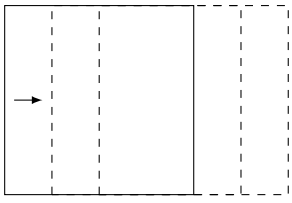
Es stehen Ihnen folgende Farben zur Verfügung:

black	gray	white	red	orange
yellow	magenta	blue	cyan	green

Die Schildkröte befindet sich während ihrer Bewegung im „Stiftmodus“ und zeichnet ihre Spur. Mit `penUp()` wird der Stift angehoben und die Spur nicht gezeichnet. Mit `penDown()` wird der Farbstift wieder auf die Ebene gesetzt.

Aufgabe 18

Schreiben Sie eine Animation, in der ein Quadrat `quad(100)` sich horizontal nach rechts bewegt.
Tipp: Benutzen Sie die Befehle `penUp()` *und* `penDown()`.



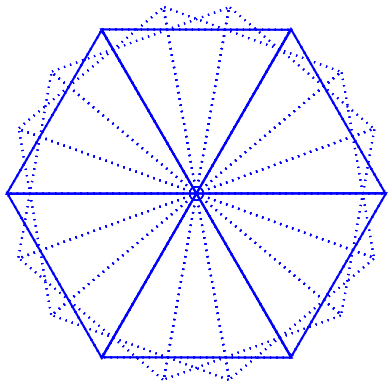
Aufgabe 19

Der Befehl `sechseck(s)` baut aus dem Befehl `dreieck(s)` ein Sechseck mit der Seitenlänge `s` und zeichnet es (Lösung zu Aufgabe 14).

Das Sechseck soll nun, wie ein Rad, um den Mittelpunkt gedreht werden. Dies erreichen Sie, indem Sie die Schildkröte (wie in Aufgabe 17) nach jedem Schritt ein wenig drehen und `sechseck(100)` erneut aufrufen.

Ergänzen Sie das Programm entsprechend.

```
from gturtle import *

def dreieck(s):
    repeat 3:
        forward(s)
        right(120)

def sechseck(s):
    repeat 6:
        dreieck(s)
        right(60)

makeTurtle()
hideTurtle()
....
```



4 Ausblick – Fraktale

Dieses Kapitel richtet sich an Anwender, die schon Programmiererfahrung haben und eine Herausforderung suchen.

Die Schildkröte zeichnet Fraktale

Ein **Fraktal** kann man als Objekt oder Muster bezeichnen, welches aus (unendlich vielen) Kopien von sich selbst aufgebaut ist. Man kann Fraktale, aufgrund ihrer Selbstähnlichkeit, mit Hilfe von *rekursiven* Funktionen zeichnen.

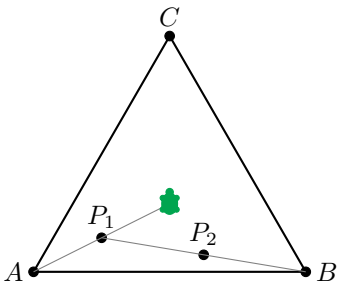
Das folgende Fraktal erzeugen wir nun aber verblüffend einfach mit Hilfe eines Zufallsprozesses:

Aufgabe 20

Erstellen Sie ein Programm nach folgender Vorgabe. Hinweise zum Code können Sie der Tabelle unten entnehmen.

Die folgenden drei Schritte sollen beliebig oft **wiederholt** werden:

1. Man wählt einen der Punkte A, B, C **zufällig** aus.
2. Der **Mittelwert** aus den Koordinaten des Zufallspunktes aus 1. und den Koordinaten der aktuellen Position der Schildkröte bilden die Koordinaten des neuen Punktes P_x.
   Beachten sie: Die Schildkröte befindet sich zu Beginn im Punkt (0,0)
3. Die Schildkröte springt zu P_x und zeichnet mit `dot(1)` einen Punkt.



Die Punkte A, B, C können Sie folgendermassen definieren:

```
A,B,C = (-300,-220),(300,-220),(0,280)
```

`from random import *`	Modul mit Befehlen für Zufallszahlen
`randint(0,n)`	liefert Zufallszahl zwischen `0` und `n`
`setPos(x,y)`	setzt Schildkröte auf die Position `(x,y)`
`dot(d)`	zeichnet gefüllten Kreis mit Durchmesser `d`

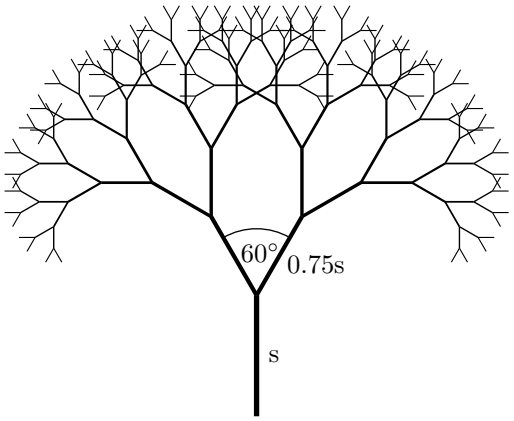
Die folgende Aufgabe ist nur für Anwender lösbar, die sich mit *Rekursion* auskennen.

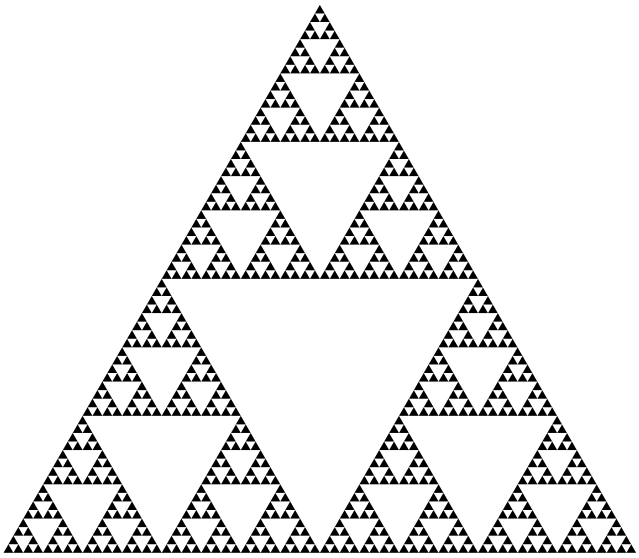
Das Ziel der Aufgabe ist es, ein weiteres Fraktal, einen sogenannten binären Baum, zu zeichnen. Er lässt sich auf eine einfache Grundfigur zurückführen. Die Funktion `tree(s)` wird jeweils im linken und rechten Ast mit einer verkürzten Seitenlänge `s` aufgerufen. Ab einer bestimmten Seitenlänge bricht der Prozess ab.

Aufgabe 21

Erstellen Sie eine Funktion `tree(s)` die den abgebildeten Baum *rekursiv* erzeugt. Die Streckenlänge `s` wird bei jeder Stufe um 25% verkürzt. Der Winkel zwischen den Ästen beträgt 60°.

Optional kann man die Astdicke pro Stufe ebenfalls um 20% dünner werden lassen.





Sierpinski-Dreieck (Lösung der Aufgabe 20)

Befehlsübersicht

`from bib import *`	importiert alle Befehle, die in `bib` definiert sind
`makeTurtle()`	erstellt das Fenster mit der Schildkröte
`forward(100)`, `fd(100)`	100 Schritte vorwärts gehen
`back(100)`, `bk(100)`	100 Schritte rückwärts gehen
`right(60)`, `rt(60)`	60 Grad nach rechts drehen
`left(60)`, `lt(60)`	60 Grad nach links drehen
`clear()`	löscht Bildschirm
`clear("blue")`	löscht Bildschirm und setzt Hintergrundfarbe auf blau
`repeat 5:`	nach `:` eingerückter Programmblock wird 5 Mal wiederholt
`penUp()`	die Schildkröte wechselt in den Wandermodus
`penDown()`	die Schildkröte wechselt in den Zeichenmodus
`setPenColor("red")`	wechselt die Stiftfarbe auf rot
`def befehl():`	erstellt einen Befehl mit Namen `befehl`
`def befehl(param):`	erstellt einen Befehl mit Namen `befehl` und Parameter `param`
`delay(5)`	die Schildkröte wartet 5 Zeiteinheiten
`hideTurtle()`	versteckt die Schildkröte
`showTurtle()`	zeigt die Schildkröte wieder an
`randint(0,n)`	liefert Zufallszahl zwischen `0` und `n`
`setPos(x,y)`	setzt Schildkröte auf die Position `(x,y)`
`dot(d)`	zeichnet gefüllten Kreis mit Durchmesser `d`.

Programmieren mit Python

Informationstechnologie und Ausbildung
ETH Zürich, CAB F 15.1
Universitätstrasse 6
CH-8092 Zürich

www.ite.ethz.ch
www.abz.inf.ethz.ch