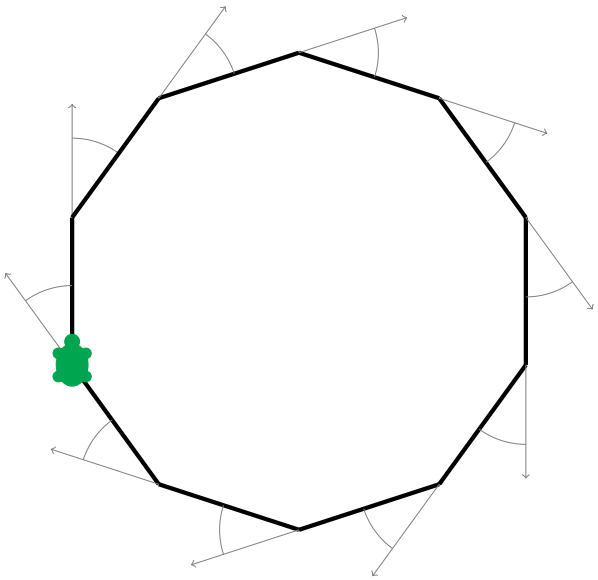
ABZ AUSBILDUNGS- UND BERATUNGSZENTRUM
FÜR INFORMATIKUNTERRICHT

Urs Hauser Juraj Hromkovič Tobias Kohn
Dennis Komm Giovanni Serafini Jacqueline Staub

Programmieren mit Python



UNIVERSITÄT TRIER

ETH zürich

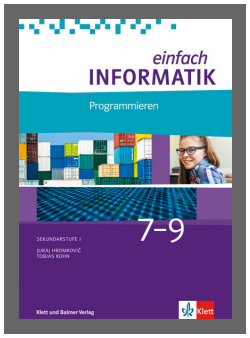
Programmierumgebung

Die vorliegenden Unterrichtsunterlagen wurden für die Programmierumgebungen XLogoOnline (`http://xlogo.uni-trier.de`), TigerJython (`http://jython.tobiaskohn.ch`) und WebTigerJython (`https://webtigerjython.ethz.ch/`) erstellt und stehen kostenlos zur Verfügung.

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 im Klett-Verlag erhältlich.

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



Programmieren

Daten darstellen, verschlüsseln, komprimieren

Strategien entwickeln

Nutzungsrechte

Die TGT stellt dieses Leitprogramm zur Förderung des Unterrichts interessierten Lehrkräften oder Institutionen zur internen Nutzung kostenlos zur Verfügung.

TGT

Die Turtle Group Trier der Universität Trier unterstützt Schulen und Lehrkräfte, die ihren Informatikunterricht entsprechend auf- oder ausbauen möchten, mit einem vielfältigen Angebot. Es reicht von individueller Beratung und Unterricht durch Universitäts-Professoren und das TGT-Team direkt vor Ort in den Schulen über Ausbildungs- und Weiterbildungskurse für Lehrkräfte bis zu Unterrichtsmaterialien.

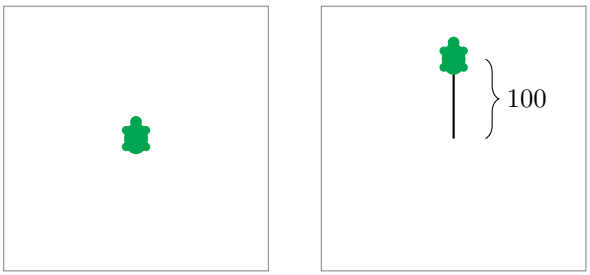
`http://xlogo.uni-trier.de/`

1 Grundbefehle

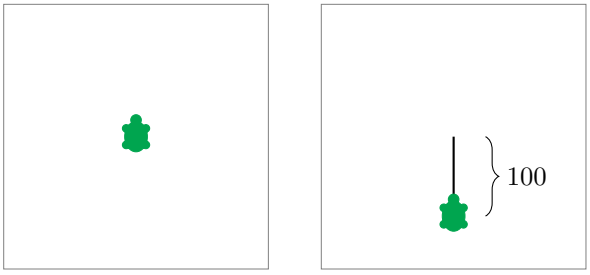
Ein **Computerbefehl** ist eine Anweisung, die der Computer versteht und ausüben kann. Der Computer kennt eigentlich nur sehr wenige Befehle und alle komplizierten Tätigkeiten, die wir vom Computer vollbracht haben wollen, müssen wir aus den einfachen Computerbefehlen zusammensetzen. Diese Folge von Computerbefehlen nennen wir **Programm**. Programme zu schreiben ist nicht immer einfach. Es gibt Programme, die aus Millionen von Befehlen zusammengesetzt sind. Hierbei die Übersicht nicht zu verlieren, erfordert ein durchdachtes und sauberes Vorgehen, das wir in diesem Programmierkurs erlernen werden.

Gerade Linien zeichnen

Mit dem Befehl `forward(100)` oder `fd(100)` befiehlst du der Schildkröte 100 Schritte nach vorne zu gehen:



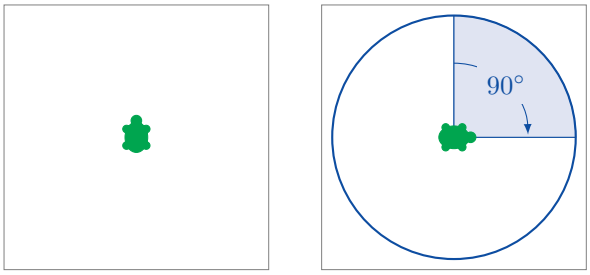
Mit dem Befehl `back(100)` oder `bk(100)` geht die Schildkröte 100 Schritte zurück:



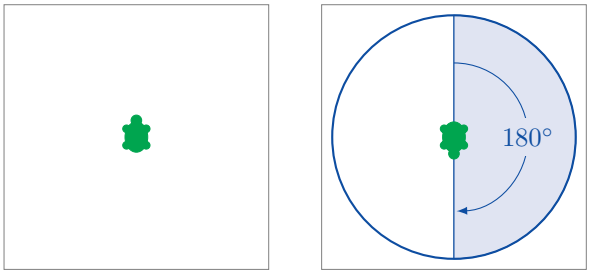
Drehen

Die Schildkröte läuft immer in die Richtung, in die sie gerade schaut.

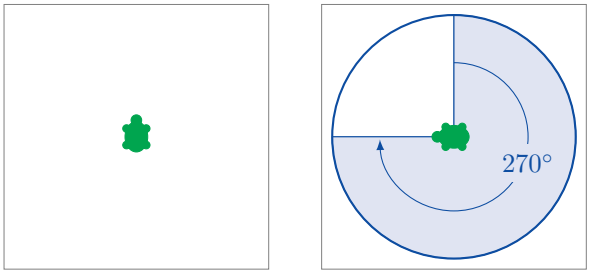
Mit dem Befehl `right(90)` oder `rt(90)` dreht sich die Schildkröte um 90° nach rechts. Dies entspricht einem Viertelkreis:



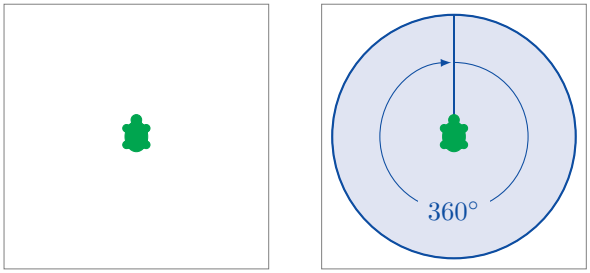
Der Befehl `right(180)` oder `rt(180)` dreht die Schildkröte um 180° nach rechts. Dies entspricht einer halben Umdrehung:



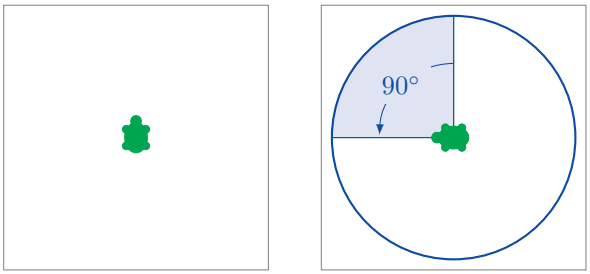
`right(270)` oder `rt(270)` drehen die Schildkröte um 270° nach rechts:



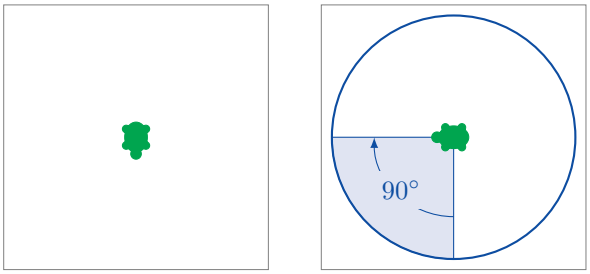
Die Befehle `right(360)` und `rt(360)` drehen die Schildkröte um 360° nach rechts. Dies entspricht einer vollen Umdrehung:



Mit dem Befehl `left(90)` oder `lt(90)` dreht sich die Schildkröte um 90° nach links:



Beachte, dass sich das Drehen nach links und rechts auf die Sicht der Schildkröte bezieht, wie das folgende Beispiel mit dem Befehl `rt(90)` zeigt:



Programmieren

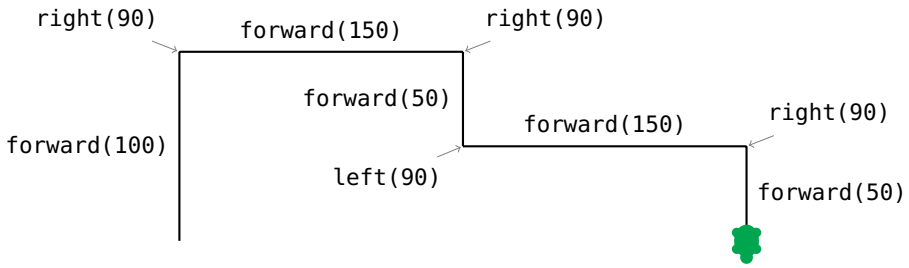
Programmieren bedeutet, eine Folge von Computerbefehlen hintereinander zu schreiben.

Aufgabe 1

Tippe das folgende Programm ab und führe es aus:

```
from gturtle import *
makeTurtle()
forward(100)
right(90)
forward(150)
right(90)
forward(50)
left(90)
forward(150)
right(90)
forward(50)
```

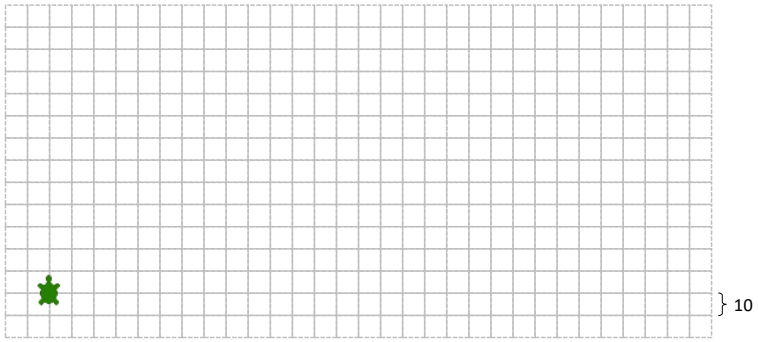
Hast du folgendes Bild gezeichnet?



Aufgabe 2

Schreibe das folgende Programm und führe es aus:

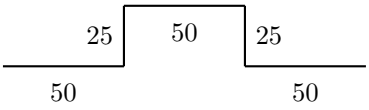
```
from gturtle import *
makeTurtle()
forward(100)
right(90)
forward(120)
right(90)
forward(80)
right(90)
forward(100)
right(90)
forward(50)
```



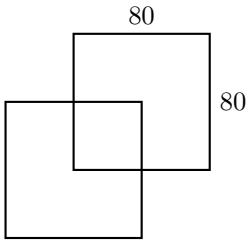
Zeichne das entstandene Bild neben das Programm oben und beschreibe (wie in Aufgabe 1) welcher Befehl was verursacht hat.

Aufgabe 3

Schreibe Programme, die die folgenden Bilder zeichnen. Bei allen Bildern darfst du die Startposition der Schildkröte selbst wählen.



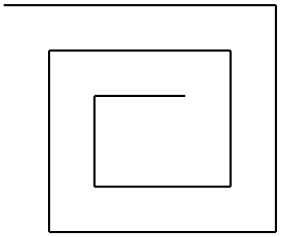
(a)



(b)

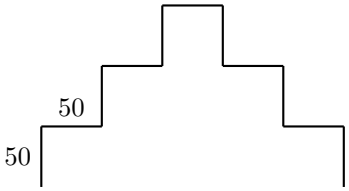
Aufgabe 4

Zeichne die folgende Spirale. Die Größe und den Startpunkt darfst du frei wählen.



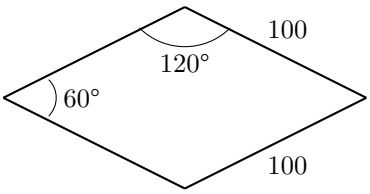
Aufgabe 5

Lino will das folgende Bild zeichnen, dabei allerdings nur die Befehle `fd` und `rt` einsetzen. Hilf ihm.



Aufgabe 6

Schreibe ein Programm, das eine Raute gemäß der folgenden Skizze zeichnet.



Stift heben und senken

Wie du inzwischen weißt, hält die Schildkröte einen Stift in der Hand und zeichnet immer, wenn sie sich bewegt. Wir nennen dies den **Stiftmodus**.

Im **Wandermodus** bewegt sich die Schildkröte auf dem Bildschirm ohne zu zeichnen. In den Wandermodus kommst du mit dem Befehl

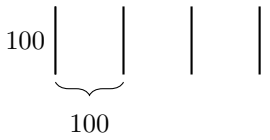
`penUp()` oder kurz `pu()`.

Aus dem Wandermodus zurück in den Stiftmodus kommst du mit dem Befehl

`penDown()` oder kurz `pd()`.

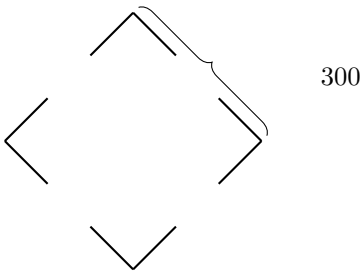
Aufgabe 7

Zeichne das folgende Bild mit einem Programm:



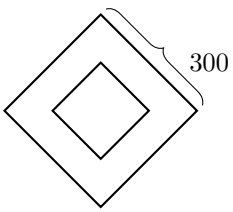
Aufgabe 8

Zeichne das folgende Bild mit einem Programm.



Aufgabe 9

(*) Zeichne das folgende Bild mit einem Programm.



2 Der Befehl `repeat`

Um ein einfaches Quadrat der Seitenlänge 100 zu zeichnen, lassen wir die zwei Befehle `forward(100)` und `right(90)` vier Mal wiederholt ausführen:

```
from gturtle import *
makeTurtle()

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

Diese langweilige und wiederholende Schreibarbeit können wir uns glücklicherweise sparen, denn die Schildkröte kann Wiederholungen dieser Art für uns übernehmen, ohne das wir dieselben Befehle immer und immer wieder eintippen müssen.

Konkret verwenden wir dazu eine **`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)
```

Die Zahl 4 hinter **`repeat`** gibt an, wie oft die folgenden Befehle ausgeführt werden sollen. Anschließend muss zwingend ein Doppelpunkt „`:`“ folgen und alle Befehle, die wiederholt werden sollen, müssen mittels <TABULATOR> gleichmäßig **eingerückt** werden.

Aufgabe 10

Kannst du den Befehl **repeat** verwenden, um das Programm kürzer zu schreiben?

```
forward(75)
left(90)
forward(75)
left(90)
forward(75)
left(90)
forward(75)
left(90)
```

Aufgabe 11

Verwende den Befehl **repeat**, um das folgende Programm zu verkürzen.

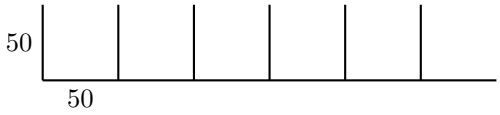
```
forward(50
right(60)
forward(50)
right(60)
forward(50)
right(60)
forward(50)
right(60)
forward(50)
right(60)
forward(50)
right(60)
```

Aufgabe 12

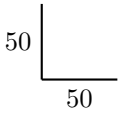
Schreibe das folgende Programm so kurz wie möglich.

```
forward(200)
left(120)
forward(200)
right(120)
forward(200)
left(120)
forward(200)
right(120)
```

Wir wollen nun das folgende Bild mit Hilfe des Befehls **repeat** zeichnen:



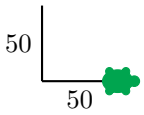
Bevor wir anfangen zu zeichnen, müssen wir uns überlegen, welches Muster mehrmals vorkommt. Wir können als zu wiederholendes Muster das folgende Bild nehmen:



Aus der unteren linken Ecke zeichnen wir das Muster mit dem folgenden Programm:

```
forward(50)
back(50)
right(90)
forward(50)
```

Nach der Ausführung dieses Programms steht die Schildkröte wie in der folgenden Abbildung gezeichnet und schaut nach rechts:



Bevor sie nun das Muster ein zweites Mal zeichnen kann, muss die Schildkröte wieder nach oben schauen, was wir mit dem Befehl `left(90)` erreichen können.

Wir setzen die beiden Teile (das *Muster* und die *Ausrichtung*) zusammen und erhalten ein funktionierendes Programm, wenn wir sämtliche Befehle 6 Mal wiederholen:

```
repeat 6:
   forward(50)
   back(50)
   right(90)
   forward(50)
   left(90)
```

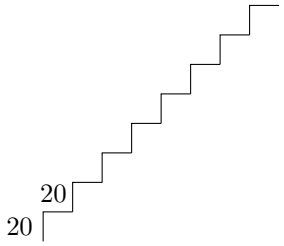
Viele Aufgaben können wir mit dieser Vorgehensweise lösen. Du musst dabei einerseits immer ein Programm für das *Muster* und andererseits ein Programm für das *Ausrichten* der Schildkröte entwickeln. Dein Programm sieht dann wie folgt aus.

```
repeat ANZAHL :
   MUSTER
   AUSRICHTEN
```

Aufgabe 13

Zeichnen von Treppen.

(a) Zeichne eine Treppe mit 8 Stufen der Höhe 20 und Breite 20:

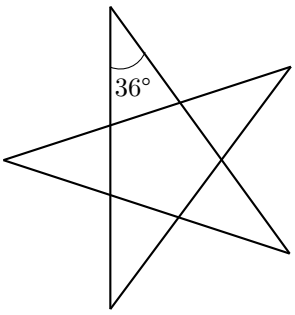


(b) Zeichne eine Treppe mit 5 Stufen der Höhe 50 und Breite 75.

(c) Zeichne eine Treppe mit 50 Stufen der Höhe 5 und Breite 3.

Aufgabe 14

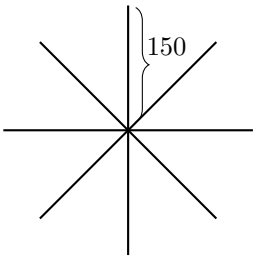
Schreibe ein Programm, um einen fünfzackigen Stern der Höhe 200 zu zeichnen:



Aufgabe 15

Nun geht es rund...

(a) Zeichne den folgenden Stern mit 8 Strahlen der Länge 150:

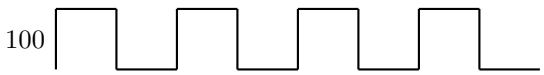


(b) Zeichne einen Stern mit 16 Strahlen der Länge 100.

(c) Zeichne einen Stern mit 360 Strahlen.

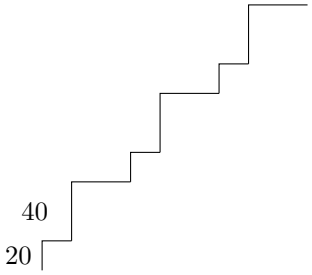
Aufgabe 16

Zeichne das folgende Muster mit so wenigen Befehlen wie möglich:



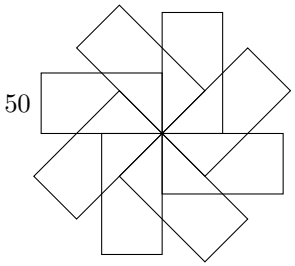
Aufgabe 17

Zeichne die folgende Treppe mit `repeat 3:`



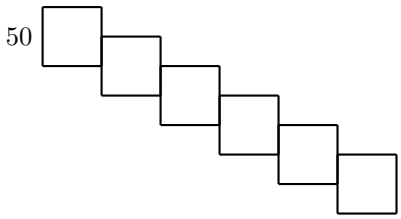
Aufgabe 18

Zeichne das folgende Muster mit `repeat`:



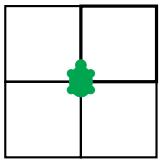
Aufgabe 19

(*) Zeichne die folgende Treppe mit `repeat`:



Repeat in Repeat

Das Fenster unten kann man ganz elegant zeichnen, indem man das Bild aus vier Quadraten zusammensetzt: Die Schildkröte steht zu Beginn in der Mitte des Fensters und zeichnet zuerst das Quadrat rechts oben. Danach dreht sie sich um 90° nach links und zeichnet das nächste Quadrat oben links usw.



Das entsprechende Programm sieht wie folgt aus. Fällt dir etwas auf?

```
repeat 4:
    forward(50)
    right(90)
left(90)
repeat 4:
    forward(50)
    right(90)
left(90)
repeat 4:
    forward(50)
    right(90)
left(90)
repeat 4:
    forward(50)
    right(90)
left(90)
```

Genau, das Programm enthält dieselben vier Zeilen immer wieder. Lass uns die mühselige Schreibarbeit sparen und stattdessen den **repeat**-Befehl einsetzen. Wie du siehst, kann ein Repeat selbst wieder ein Repeat enthalten. Man nennt diese Schleife *verschachtelt*.

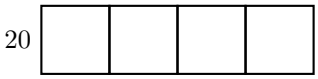
```
repeat 4:
    repeat 4:
       forward(50)
       right(90)
    left(90)
```

Vorsicht bei der Einrückung: Dass `left` hier nur einfach eingerückt ist, während `forward` und `right` doppelt eingerückt sind, ist keine Willkür. Die Linksdrehung findet nur 4 mal statt, während andererseits 16 Vorwärtsbewegungen ausgeführt werden.

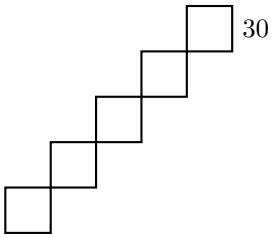
Aufgabe 20

Zeichne die folgenden Bilder mit einem verschachtelten **`repeat`**:

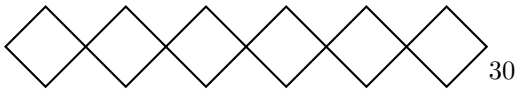
(a)



(b)



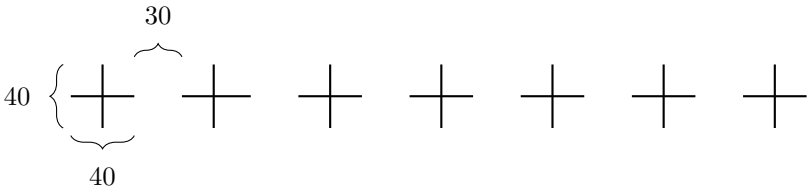
(c)



(d)

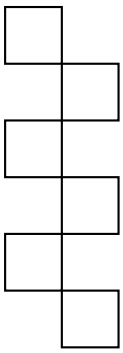
Aufgabe 21

Schreibe ein Programm für das folgende Bild mit möglichst wenigen Befehlen:



Aufgabe 22

(*) Zeichne das folgende Muster mit möglichst wenigen Befehlen:



3 Aus kleinen Bausteinen größere bauen

Manche Programme enthalten viele identische Figuren und sind daher mühsam und aufwendig zu schreiben und schwierig zu lesen. Darum führen wir den Befehl **def** ein, mit welchem häufig verwendeten Programmen ein Name zugewiesen werden kann. Um beispielsweise einen neuen Befehl `quadrat100()` zu definieren, gehen wir wie folgt vor:

```
from gturtle import *

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

makeTurtle()
```

Bei der Ausführung per Playbutton 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 den neuen Befehl 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.

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

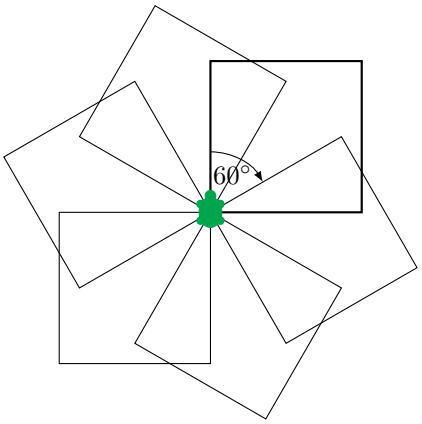
Eigene Befehle sind besonders nützlich, da sie häufig verwendet werden. Dies ist beispielsweise hier der Fall, wo wir den neuen Befehl `quadrat100()` auf diese Weise bequem sechsmal aufrufen können:

```
from gturtle import *

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

makeTurtle()

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



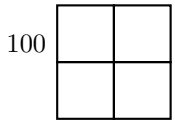
Wir können eine beliebige Anzahl von Befehlen definieren und ausführen lassen.

Aufgabe 23

Nutze den Befehl `quadrat100()`, um das folgende Bild zu zeichnen:

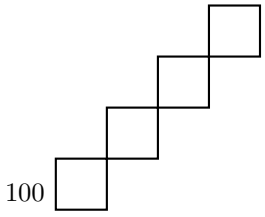
Aufgabe 24

Nutze den Befehl `quadrat100()`, um das folgende Bild zu zeichnen:



Aufgabe 25

Nutze den Befehl `quadrat100()`, um das folgende Bild zu zeichnen:



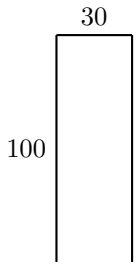
Aufgabe 26

Manfred hat einen neuen Befehl definiert mit dem Namen `lakula()`. Dieser Befehl erlaubt es, die rechts abgebildete Form zu zeichnen:

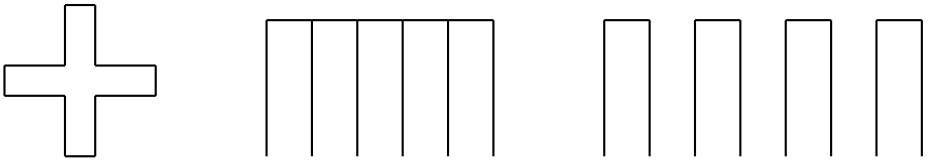
```
from gturtle import *

def lakula():
   forward(100)
   right(90)
   forward(30)
   right(90)
   forward(100)

makeTurtle()
lakula()
```



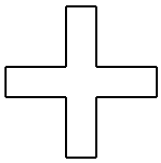
Verwende den Befehl `lakula()`, um die folgenden drei Figuren zu zeichnen:



(a) (b) (c)

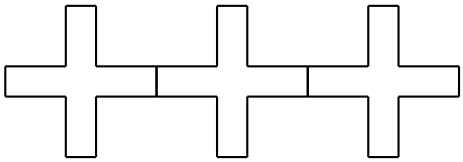
Aufgabe 27

Definiere den Befehl `kreuz()` um die Figur aus Aufgabe 26a abzuspeichern. Markiere im Bild unten den Start- und Endpunkt der Schildkröte sowie deren Ausrichtung.



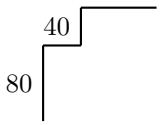
Aufgabe 28

Benutze nun den Befehl `kreuz()`, um die folgende Figur zu zeichnen:



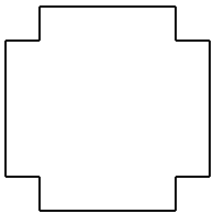
Aufgabe 29

Schreibe ein Programm `haken()`, um die folgende Figur zu zeichnen:

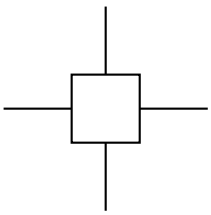


Aufgabe 30

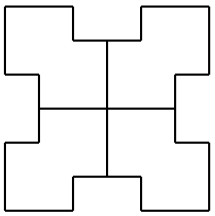
Benutze das Programm `haken()`, um folgende Figuren zu zeichnen:



(a)



(b)



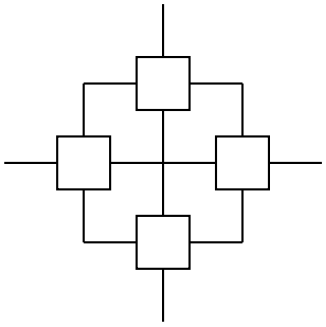
(c)

Aufgabe 31

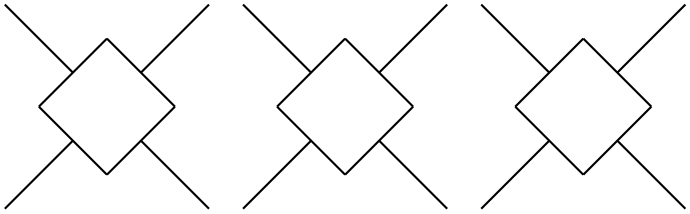
Schreibe nun ein Programm `stern()`, welches die Figur aus Aufgabe 33 (b) zeichnet.

Aufgabe 32

Benutze das Programm `stern()`, um folgende Figuren zu zeichnen:



(a)



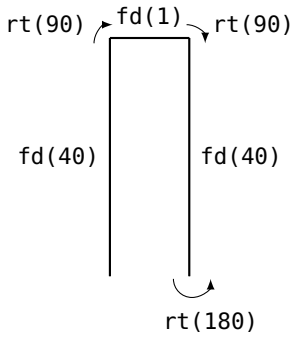
(b)

Dicke Linien und schwarze Quadrate

Aufgabe 33

Alex zeichnet eine fette Linie und benutzt dazu das folgende Programm:

```
fd(40)
rt(90)
fd(1)
rt(90)
fd(40)
rt(180)
```



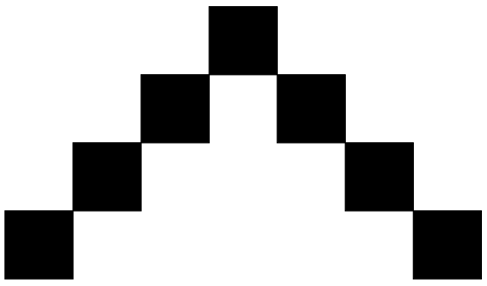
Die fette Linie entsteht, indem man zwei Linien so dicht nebeneinander zeichnet, dass man die Lücke dazwischen nicht mehr sieht. Verwende den Befehl `repeat`, um anstatt nur zwei Linien jetzt 40 Linien nebeneinander zu zeichnen. Speichere deine Lösung anschließend als neuen Befehl mit dem Namen `fett40()` ab.

Aufgabe 34

Nutze `fett40()` um das folgende Bild zu zeichnen:

Aufgabe 35

Verwende das Programm `fett40()`, um das folgende Bild zu zeichnen:



Arbeiten mit Farben

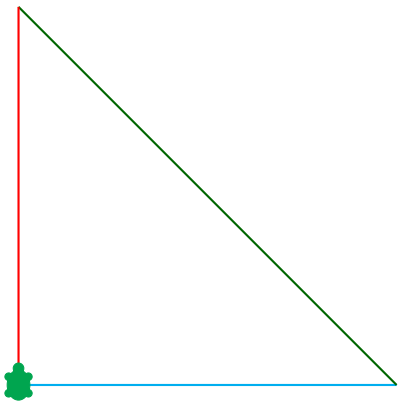
Die Schildkröte kann mit verschiedenen Farben zeichnen. Du musst ihr einfach angeben, welche Farbe sie für den Stift verwenden soll. Es existieren über hundert verschiedene Farben, die du in Englisch angibst. Eine Übersicht einiger nützlicher Farben findest du in der folgenden Tabelle:

black	gray	white	red	orange
yellow	magenta	blue	cyan	green

Mit dem Befehl `setPenColor("red")` kann die Stiftfarbe beispielsweise auf rot geändert werden. Achte darauf, die Farbe in Anführungszeichen zu setzen.

Ein Beispiel:

```
setPenColor("red")
fd(100)
rt(135)
setPenColor("green")
fd(141)
rt(135)
setPenColor("cyan")
fd(100)
rt(90)
```



Aufgabe 36

Zeichne eine Deutsche Flagge in der Größe 240x360:

Aufgabe 37

Zeichne die folgenden Flaggen:

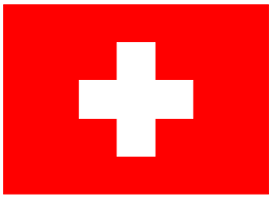
(a)

(b)

(c)

Aufgabe 38

(*) Zeichne eine Schweizer Flagge und eine Griechenland-Flagge:

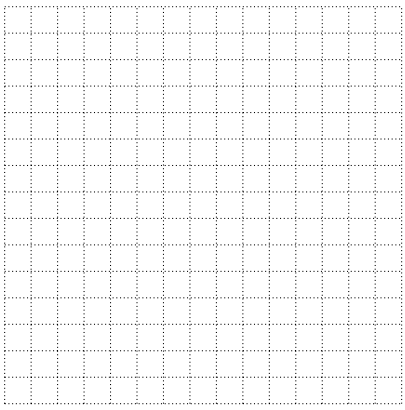


(a)

(b)

Aufgabe 39

Projektaufgabe: Kreiere deine eigene Pixelart Figur und programmiere sie.



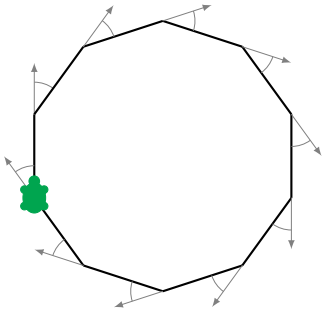
4 Regelmäßige Vielecke und Kreise

Regelmäßige Vielecke

Ein regelmäßiges k-Eck hat k Ecken und k gleich lange Seiten. Wenn du ein Vieleck, zum Beispiel ein 10-Eck, mit Bleistift zeichnen möchtest, musst du 10 Linien zeichnen und nach jeder Linie „ein bisschen“ die Richtung ändern (drehen).

Wie viel muss man drehen?

Wenn man ein Vieleck zeichnet, dreht man mehrmals unterwegs, aber am Ende steht man genau an der gleichen Stelle und schaut in genau die gleiche Richtung wie am Anfang.



Das bedeutet, dass man sich unterwegs volle 360° gedreht hat. Wenn man also ein regelmäßiges 10-Eck zeichnet, hat man sich genau zehn Mal gedreht, und zwar immer um einen gleich großen Winkel. Dieser Winkel ist:

$$\frac{360°}{10} = 36°$$

Daher muss man sich immer um 36° drehen: `rt 36`. Probieren wir das aus, indem wir folgendes Programm schreiben:

```
from gturtle import *
makeTurtle()

repeat 10:
    forward(50)
    right(36)
```

Aufgabe 40

Zeichne die folgenden regelmäßigen Vielecke:

(a) ein 5-Eck mit der Seitenlänge 180,

(b) ein 12-Eck mit der Seitenlänge 50,

(c) ein 4-Eck mit der Seitenlänge 200,

(d) ein 6-Eck mit der Seitenlänge 100,

(e) ein 3-Eck mit der Seitenlänge 200 und

(f) ein 18-Eck mit der Seitenlänge 20.

Wenn man ein 7-Eck zeichnen will, hat man das Problem, dass man 360 nicht ohne Rest durch 7 teilen kann. In diesem Fall lässt man das Resultat durch den Computer ausrechnen, indem man

```
360/7
```

schreibt („`/`“ bedeutet für den Computer „teile“). Der Computer findet dann das genaue Resultat. Somit kann man ein 7-Eck mit Seitenlänge 100 wie folgt zeichnen:

```
repeat 7:
    forward(100)
    right(360/7)
```

Probiere es aus.

Kreise zeichnen

Mit den Befehlen `fd` und `rt` kann man keine genauen Kreise zeichnen. Wie du aber sicherlich beobachtet hast, sehen Vielecke mit vielen Ecken Kreisen sehr ähnlich. Wenn wir also viele Ecken und sehr kurze Seiten nehmen, erhalten wir dadurch Kreise.

Aufgabe 41

Teste die folgenden vier Programme und erkläre die optischen Unterschiede.

`repeat 360:`
`  forward(1)`
`  right(1)`
	`repeat 360:`
	`  forward(1)`
	`  left(1)`
		`repeat 180:`
		`  forward(2)`
		`  right(2)`
			`repeat 360:`
			`  forward(3.5)`
			`  right(1)`

Tipp: 3.5 bedeutet $3\frac{1}{2}$.

Aufgabe 42

Versuche, die folgenden Halbkreise zu zeichnen. Die Größen darfst du selber bestimmen:

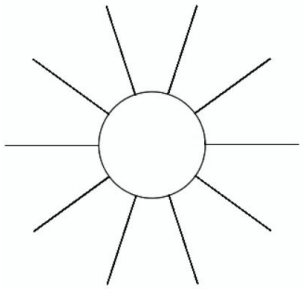
(a)

(b)

Schaffst du es auch, folgende Reihe von Halbkreisen zu zeichnen?

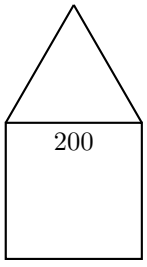
Aufgabe 43

(*) Zeichne die Sonne. Du kannst dir die Länge der Strahlen und den Umfang des Kreises selbst aussuchen.

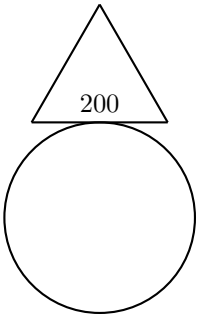


Aufgabe 44

Nutze deine neuen Erfahrungen, um die folgenden Bilder zu zeichnen. Die Größe des Kreises darfst du selber wählen:



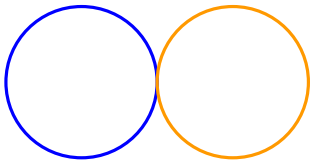
(a)



(b)

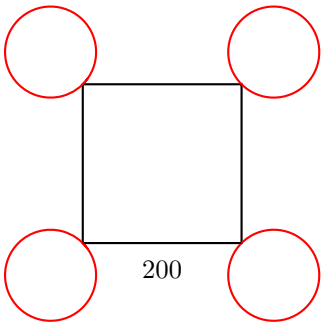
Aufgabe 45

Zeichne das folgende Bild. Die Schildkröte ist am Anfang im Schnittpunkt der beiden Kreise:



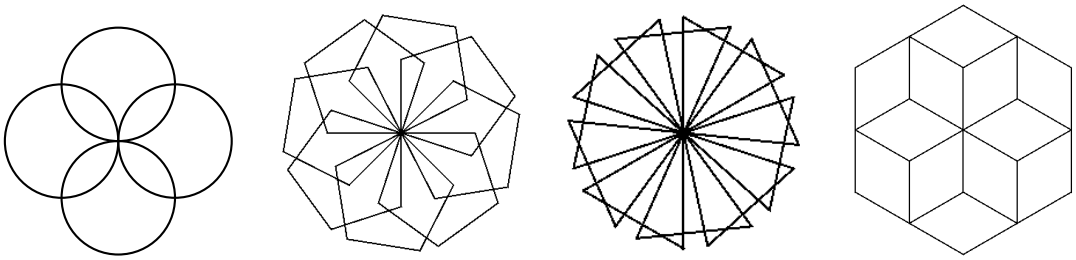
Aufgabe 46

Schreibe ein Programm zum Zeichnen des folgenden Bildes. Die Kreisgröße darfst du selber wählen:



Aufgabe 47

(*) Zeichne die folgenden Mandalas:



(a) (b) (c) (d)

Überlege dir eigene Mandalas und programmiere sie.

5 Programme mit Parametern

In Kapitel 3 haben wir gelernt, Programmen einen Namen zu geben und sie dann mit dem Namen aufzurufen, um das gewünschte Bild vom Computer zeichnen zu lassen. In Kapitel 4 haben wir gelernt, Vielecke zu zeichnen. Es ist sehr aufwendig, dass wir für jedes Vieleck mit einer neuen Anzahl von Ecken ein neues Programm schreiben müssen.

Betrachten wir zum Beispiel die folgenden drei Programme:

```
repeat 7 :
    forward(50)
    right(360/ 7 )
```

```
repeat 12 :
    forward(50)
    right(360/ 12 )
```

```
repeat 18 :
    forward(50)
    right(360/ 18 )
```

Die Programme sind sich sehr ähnlich und unterscheiden sich nur in den gelben Zahlen `7`, `12` und `18`. Diese Zahlen bestimmen die Anzahl der Ecken. Wir wollen nun ein Programm schreiben, mit dem wir alle möglichen Vielecke zeichnen können:

```
def VIELECK(ECK):
    repeat ECK:
        forward(50)
        right(360/ECK)
```

Was haben wir gemacht? Überall, wo die Anzahl der Ecken im Programm steht, schreiben wir statt der Zahl einen Namen, in diesem Fall `ECK`. Damit der Computer von vornherein weiß, dass wir die Anzahl der Ecken später frei wählen wollen, muss nach dem Namen des Programms auch `ECK` geschrieben werden.

Wenn man den Befehl `VIELECK(12)` schreibt, setzt der Computer im Programm

```
repeat 12
       ECK:
    forward(50)
    right(360/ECK)
```

überall, wo `ECK` steht, die Zahl `12` ein und zeichnet so ein 12-Eck.
Probiere das Ganze mit 3, 4, 5 und 6 für `ECK` aus.

Wir nennen `ECK` einen **Parameter**. Im Beispiel oben sind 3, 4, 5 und 6 **Werte des Parameters** `ECK`. Der Computer erkennt den Parameter an den runden Klammern `()`. Deshalb müssen alle Parameter in den Klammern hinter dem Namen des Programms stehen.

Aufgabe 48

Die folgenden Programme zeichnen Quadrate verschiedener Seitenlängen:

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

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

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

Die gelben Zahlen 100, 50, 200 kann man als Werte eines Parameters betrachten, die die Größe des Quadrats bestimmen. Schreibe ein Programm mit einem Parameter `GR`, mit welchem beliebig große Quadrate gezeichnen werden können:

```
def QUADRAT(GR):
    ...
```

Aufgabe 49

Die folgenden Programme zeichnen unterschiedlich große Kreise:

```
repeat 360:
    forward( 1 )
    right(1)
```

```
repeat 360:
    forward( 12 )
    right(1)
```

```
repeat 360:
    forward( 3 )
    right(1)
```

Schreibe ein Programm mit einem Parameter, mit dem man beliebig große Kreise zeichnen kann und probiere es dann für die Parameterwerte 1, 2, 3, 4 und 5 aus. Den Namen des Programms und den Namen des Parameters darfst du dir selber aussuchen.

Aufgabe 50

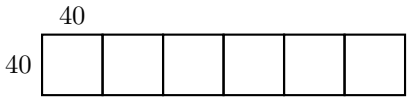
Schreibe ein Programm mit einem Parameter, mit welchem beliebig große gleichseitige Dreiecke gezeichnet werden können. Zeichne dann mit diesem Programm nacheinander Dreiecke der Größen

20, 40, 60, 80, 100, 120, 140, 160 und 180.

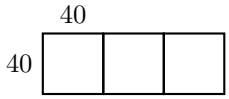
Was entsteht dabei?

Aufgabe 51

Wir wollen nun Quadrate der Seitenlänge 40 nebeneinander zeichnen. Schreibe ein Programm `QUADRATE` mit einem Parameter `ANZ`. Der Parameter `ANZ` soll die Anzahl der Quadrate bestimmen. Wenn man also `QUADRATE`(6) aufruft, soll die Schildkröte das folgende Bild zeichnen



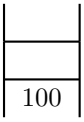
wohingegen `QUADRATE`(3) das folgende Bild erzeugt:



Aufgabe 52

Schreibe ein Programm `leiter(ANZ)`, um Leitern mit beliebig viele Sprossen zu zeichnen. Der Parameter `ANZ` gibt an, wie viele Sprossen die Leiter hat.

(a) leiter(1)



(b) leiter(2)

(c) leiter(3)

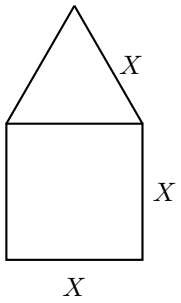
Aufgabe 53

Schreibe ein Programm mit einem Parameter, das ausgefüllte Quadrate beliebiger Größe zeichnen kann.

Tipp: Du kannst zuerst ein Programm für ein ausgefülltes Quadrat der Größe 100x100 und ein Programm für ein ausgefülltes Quadrat der Größe 50x50 schreiben, um zu erkennen, wo der Parameter eingesetzt werden muss.

Aufgabe 54

Schreibe ein Programm mit einem Parameter `X`, das beliebig große Häuser wie in der folgenden Abbildung zeichnet.



Programm mit mehreren Parametern

Ein Programm kann mehr als einen Parameter haben. Wenn wir Vielecke zeichnen, können wir einen Parameter `ECK` für die Anzahl der Ecken und einen Parameter `GR` für die Seitenlänge bestimmen.

In den folgenden Programmen ist der Parameter `ECK` mit gelb und der Parameter `GR` mit grün markiert:

```
repeat 13 :
  forward( 100 )
  right(360/ 13 )
```

```
repeat 3 :
  forward( 300 )
  right(360/ 3 )
```

```
repeat 17 :
  forward( 10 )
  right(360/ 17 )
```

```
repeat 60 :
  forward( 3 )
  right(360/ 60 )
```

Damit können wir jetzt ein Programm für unterschiedliche Vielecke schreiben:

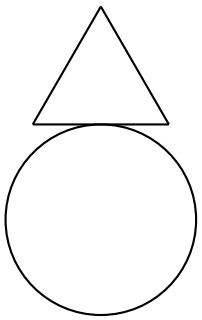
```
def VIELECKE(ECK, GR):
    repeat ECK:
        forward(GR)
        right(360/ECK)
```

Teste das Programm `VIELECKE` mit den folgenden Aufrufen:

```
VIELECKE (12, 60), VIELECKE (12, 45), VIELECKE (8, 30), VIELECKE (9, 30),
VIELECKE (7, 31), VIELECKE (11, 50)
```

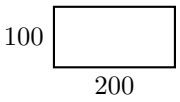
Aufgabe 55

Schreibe ein Programm mit zwei Parametern und zeichne damit das folgende Bild. Dabei soll die Kreisgröße, sowie die Größe des Dreiecks frei wählbar sein.



Aufgabe 56

Das rechtsstehende Programm zeichnet ein Rechteck der Breite 200 und der Höhe 100.

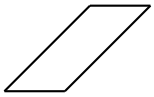


Überprüfe es und schreibe ein Programm mit zwei Parametern, sodass Rechtecke beliebiger Breite und Höhe gezeichnet werden können.

```
forward(100)
right(90)
forward(200)
right(90)
forward(100)
right(90)
forward(200)
```

Aufgabe 57

Das rechtsstehende Programm zeichnet ein Parallelogramm.

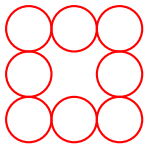


Schreibe ein Programm mit zwei Parametern, welches solche Parallelogramme mit beliebigen Seitenlängen zeichnen kann.

```
repeat 2:
    right(45)
    forward(200)
    right(45)
    forward(100)
    right(90)
```

Aufgabe 58

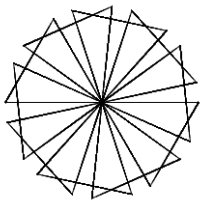
Schreibe ein Programm `kreise(ANZ, FARBE)`, um farbige Kreise in einem Quadrat anzuordnen. Die Parameter `ANZ` und `FARBE` geben die Anzahl Kreise pro Seite und die Farbe des Musters vor.



$kreise(3, \text{"}red\text{"})$

Aufgabe 59

Schreibe ein Programm `MANDALA (ANZ, ECK)`, um Mandalas aus Vielecken zu zeichnen. Mit ANZ gibst du an, wie viele Vielecke gezeichnet werden sollen, während ECK angibt, wie viele Ecken das Vieleck haben soll. Das Programm `MANDALA (10, 3)` zeichnet also das folgende Mandala:

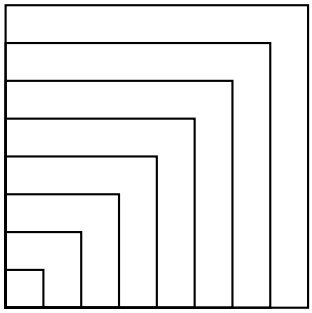


6 Programme mit Variablen

Im letzten Kapitel haben wir gesehen, wie man Programme schreibt, die einen Parameter benutzen. Wir haben hier beispielsweise gelernt, wie wir ein Quadrat zeichnen können, dessen Seitenlänge durch einen Parameter `GR` bestimmt wird.

```
def QUADRAT(GR):
    repeat 4:
        forward(GR)
        right(90)
```

Nehmen wir nun an, wir wollten 8 Quadrate mit jeweils verschiedenen Größen zeichnen, die eine gemeinsame Ecke unten links besitzen. Dabei soll das kleinste Quadrat eine Seitenlänge von 10 besitzen, das zweite eine von 20, das dritte eine von 30 usw.



Ein solches Bild können wir beispielsweise mit dem folgenden Programm `ACHTQUADRATE` erstellen, das einfach unser Programm `QUADRAT` 8-mal als Unterprogramm aufruft und dabei jeweils einen anderen Wert für `GR` übergibt.

```
def ACHTQUADRATE():
    QUADRAT(10)
    QUADRAT(20)
    QUADRAT(30)
    QUADRAT(40)
    QUADRAT(50)
    QUADRAT(60)
    QUADRAT(70)
    QUADRAT(80)
```

Diese Lösung ist nicht zufriedenstellend; stell dir vor, du müsstest ein ähnliches Programm TAUSENDQUADRATE schreiben, um 1000 Quadrate auf diesselbe Art zu zeichnen. Um Tipparbeit zu sparen, würde sich wieder anbieten, den **repeat**-Befehl zu benutzen. Die im Folgenden vorgestellte Idee besteht darin, dem Unterprogramm QUADRAT einen Parameter VARGR zu übergeben, dessen Wert sich im Verlauf des Programms ändert:

Definiere einen Parameter VARGR und gib ihm zunächst den Wert 10.
QUADRAT VARGR
Erhöhe VARGR um 10
} 8-mal

Die zentrale Frage ist, wie wir das Definieren und Erhöhen von VARGR realisieren können. Bislang haben wir Parametern ausdrücklich einen Wert zugewiesen. Jetzt wollen wir diesen Wert während der Ausführung vom Computer ändern lassen. Wir sprechen deswegen in diesem Zusammenhang von einer **Variable**. Um beispielsweise den Wert einer Variable x auf 10 zu setzen, geben wir

```
x = 10
```

ein. Wir sagen, dass wir der Variable x einen Wert *zuweisen*. Beachte, dass die Variable x nicht bereits existieren muss. Gibt es eine solche Variable noch nicht, so wird sie automatisch erzeugt und der Wert (hier 10) wird ihr zugewiesen. Wir können für das Zuweisen nicht nur konkrete Zahlen, sondern auch andere Variablen benutzen. Es ist sogar möglich, komplexere **arithmetische Ausdrücke** zu benutzen, also Rechnungen wie beispielsweise hier:

```
x = y + 9
```

Dies weist x den Wert von y zu und addiert 9. Wir können hierdurch also den Wert einer Variable vergrößern oder verringern. Wenn x zum Beispiel anfangs den Wert 7 besitzt, so ist dieser nach dem Ausführen von x=x+9 auf 16 gesetzt. Der alte Wert 7 wurde also *überschrieben*. Der Computer kümmert sich darum, dass die rechte Seite zunächst korrekt berechnet wird, und weist das Ergebnis danach x zu. Wir halten an dieser Stelle noch einmal fest, dass die Werte von Variablen sich im Laufe des Programms ändern können. Sie müssen hierzu nicht in der **def**-Zeile des Programms definiert werden, sondern können im Programm „erzeugt“ werden. Für Variablen, die ihren Wert *nicht ändern*, verwenden wir weiterhin auch die Bezeichnung **Parameter**.

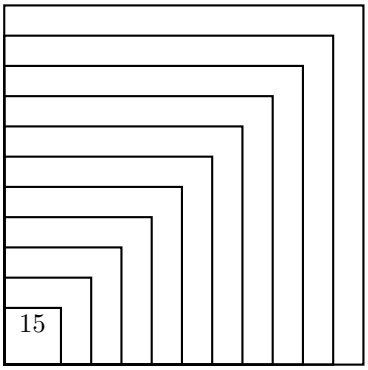
Jetzt können wir unser Programm ACHTQUADRATE schreiben, indem wir dem Unterprogramm QUADRAT die Variable VARGR übergeben, deren Wert wir nach jeder Wiederholung vergrößern. Zu Beginn definieren wir VARGR und setzen ihren Wert auf 10.

```
def ACHTQUADRATE(VARGR):
    repeat 8:
        QUADRAT(VARGR)
        VARGR = VARGR+10
```

Führe dieses Programm zunächst aus und überprüfe, ob alles richtig dargestellt wird.

Aufgabe 60

Schreibe ein Programm `ELFQUADRATE`, das ähnlich wie `ACHTQUADRATE` funktioniert, aber 11 Quadrate zeichnet, deren Seitenlänge jeweils um 8 wächst, wobei das kleinste Quadrat eine Seitenlänge von 15 besitzt.



Aufgabe 61

Schreibe ein Programm `QUADRATE` mit einem Parameter `ANZ`, das `ANZ` Quadrate zeichnet, wobei das kleinste Quadrat eine Seitenlänge von 20 besitzt und sich die Seitenlängen aufeinander folgender Quadrate um 5 unterscheiden.

Aufgabe 62

Erweitere `QUADRATE(ANZ)` zu einem Programm `QUADRATE2(ANZ,START)`, bei dem du zusätzlich die Seitenlänge des kleinsten Quadrates mit einem Parameter `START` angeben kannst.

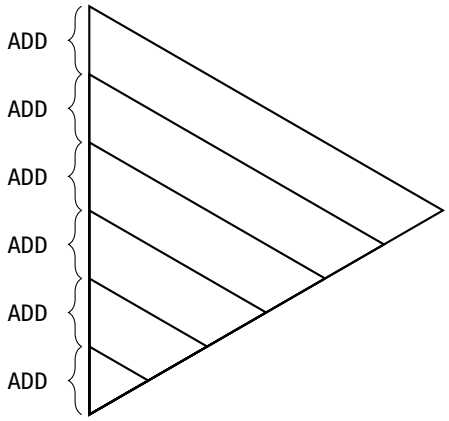
In Kapitel 4 haben wir gelernt, wie wir regelmäßige Vielecke zeichnen können. Hierbei werden 360° auf die einzelnen Drehungen verteilt, sodass die Schildkröte am Ende wieder auf ihrer Ausgangsposition steht. Das folgende Programm zeichnet ein Dreieck.

```
def DREIECK(GR):
    repeat 3:
        forward(GR)
        right(120)
```

Jetzt wollen wir Bilder aus verschiedenen Dreiecken zusammensetzen.

Aufgabe 63

Schreibe ein Programm SECHSDREIECKE, das das folgende Bild zeichnet, das aus 6 Dreiecken besteht. Die Seitenlänge soll dabei jeweils um ADD wachsen, wobei ADD ein frei wählbarer Parameter ist, der dem Programm übergeben wird. Das kleinste Dreieck soll ausserdem eine Seitenlänge von ADD besitzen.

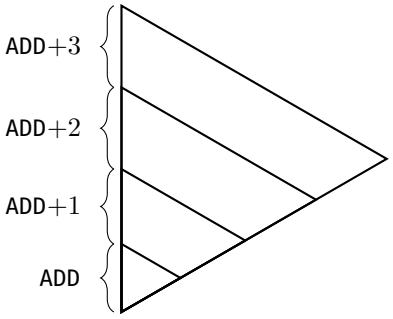


Analog zu den Programmen, die mehrere Quadrate zeichnen, soll SECHSDREIECKE DREIECK als Unterprogramm verwenden.

Aufgabe 64

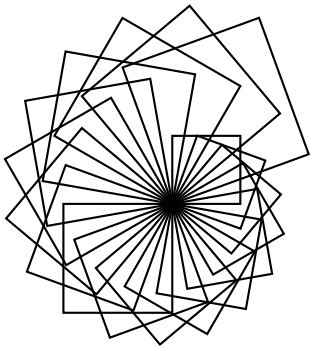
Erweitere SECHSDREIECKE zu einem Programm DREIECKE, bei dem die Anzahl der Dreiecke mit einem weiteren Parameter ANZ angegeben werden kann. Außerdem sollen die Dreicke nicht mehr konstant um ADD wachsen, sondern wie folgt. Das erste Dreieck besitzt eine Seitenlänge von ADD, die des zweiten Dreiecks ist um ADD+1 größer, die des dritten um ADD+2 usw.

Wird DREIECKE also mit dem Wert 4 für ANZ aufgerufen, soll dies zu folgender Darstellung führen.



Aufgabe 65

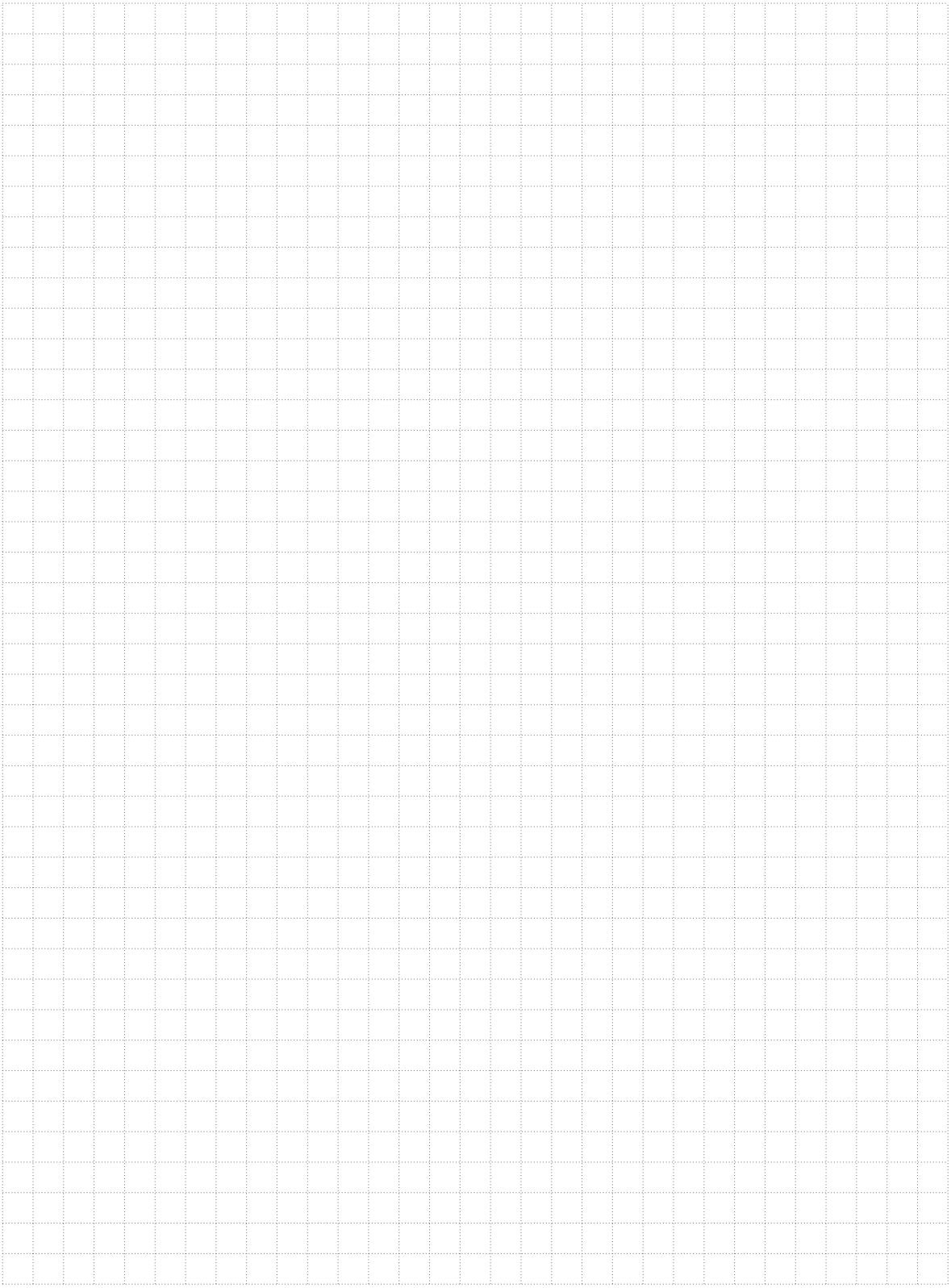
Das folgende Bild besteht aus Quadraten, die größer werden, wobei der Winkel zwischen zwei aufeinanderfolgenden Quadraten immer gleich groß ist, sodass sie gleichmäßig auf einem Kreis verteilt sind.



Erstelle ein Programm `ROTQUAD`, das derartige Bilder zeichnet. Hierbei sollen die Größe des kleinsten Quadrates, die additive Änderung, um die sich die Quadrate vergrößern, und die Anzahl der Quadrate frei wählbar sein. Die Quadrate sollen gleichmäßig auf die ganze Drehung der Schildkröte verteilt werden.

Hinweis: Berechne zuerst aus der Anzahl der Quadrate, um wie viel Grad du die Schildkröte in jedem Schritt drehen musst, indem du 360 durch diese Anzahl teilst. Verwende das Programm `QUADRAT` als Unterprogramm.

Meine Notizen



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`.