3. Der modulare Entwurf

Lernziele:

- Du lernst, wie du mit dem modularen Entwurf gut strukturierte Programme entwickeln kannst.
- Du weisst, wie du mit eigenen Befehlen die Programmiersprache erweitern kannst.

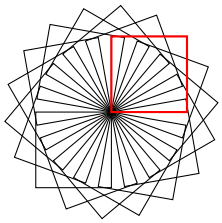
Die Grafiken der Einstiegsübung sind aus vielen identischen Bausteinen (Vielecke, Sterne) aufgebaut. Anstatt diese Bausteine immer neu zu programmieren, ist es effizienter, sie einmal zu erstellen und dann beliebig oft im Programm zu verwenden.

Ähnlich wie ein Stempel, den man einmal herstellt und dann immer wieder benutzt.

3.1 Eigene Befehle definieren

Beispiel 3.1

Die Grafik (rechts) besteht aus 18 Quadraten, die um die linke untere Ecke um 20° im Uhrzeigersinn gedreht sind:



Wir definieren für den Baustein Quadrat einen eigenen **Befehl**:

```
1  from gturtle import *
2
3  def quadrat():
4      repeat 4:
5          fd(20)
6          rt(90)
7
8  #################
9  makeTurtle()
10 ht()
11
12 repeat 18:
13     quadrat()
14     rt(20)
```

Mit **def** wird der neue Befehl definiert. Den Namen kannst du selber wählen. Wichtig sind die runden Klammern und der Doppelpunkt.

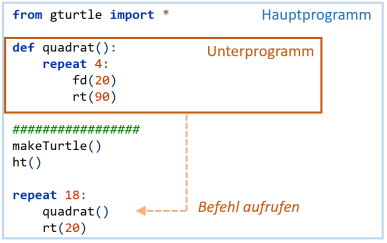
Der Körper des Befehles wird eingerückt. Er beschreibt, was ausgeführt werden soll, wenn der Befehl aufgerufen wird.

Aufruf des Befehls. Erst jetzt zeichnet der Computer.

Begriffe

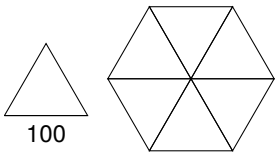
Mit **def** definieren wir einen neuen **Befehl** mit zwei runden Klammern und einem Doppelpunkt am Ende. Eingerückt folgt die Beschreibung des Programmblocks, der **Körper des Befehls**, der ausgeführt wird, wenn der Befehl aufgerufen wird.

- Befehle werden nach der Importzeile der Bibliothek definiert.
- Die Definition alleine bewirkt nichts, denn der Befehl muss im Hauptprogramm aufgerufen werden, damit er ausgeführt wird!
- Befehle werden auch als Unterprogramm des umschliessenden Hauptprogramms bezeichnet.



Aufgabe 3.1

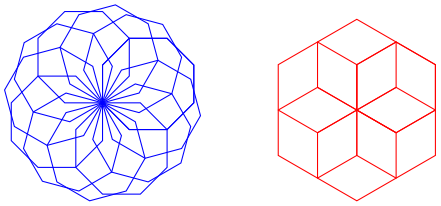
Erstelle einen Befehl `dreieck()`, der ein regelmässiges Dreieck mit einer Seitenlänge von 100 erstellt. Verwende diesen Befehl, um das Sechseck (b) zu zeichnen:



(a) (b)

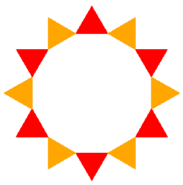
Aufgabe 3.2

Schreibe je ein Programm für die Grafiken. Definiere für die Bausteine der Grafiken eigene Befehle, wobei du die Seitenlängen und Farben selber wählen kannst.



Aufgabe 3.3

Erstelle die Grafik (a) oder eine der abgebildeten Flaggen (b). Definiere eigene Befehle für die Grundbausteine.



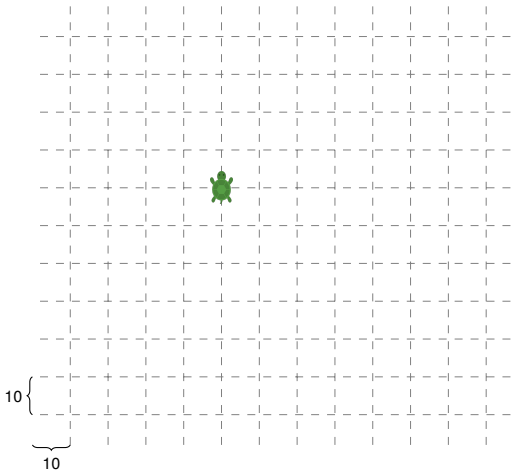
(a)

(b)

Aufgabe 3.4

Zeichne die Ausgabe des Programms ins Gitter.

```
1  from gturtle import*
2
3  def teil():
4      repeat 3:
5          fd(30)
6          rt(90)
7      lt(180)
8
9  ###################
10
11 makeTurtle()
12 ht()
13 repeat 4:
14     teil()
```



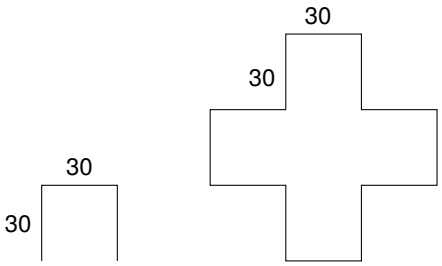
3.2 Befehle in Befehlen verwenden

Du kannst in einer Definition eines Befehls auch bereits definierte Befehle verwenden.

Beispiel 3.2

Im Beispiel wird mit `teil()` ein Arm eines Kreuzes definiert, der dann im Befehl `kreuz()` in Zeile 10 wieder verwendet wird. Das Kreuz als Ganzes haben wir hier dann gedreht.

```
1  from gturtle import *
2
3  def teil():
4      repeat 3:
5          fd(30)
6          rt(90)
7
8  def kreuz():
9      repeat 4:
10         teil()
11         rt(180)
12
13 ##################
14 makeTurtle()
15
16 kreuz()
```

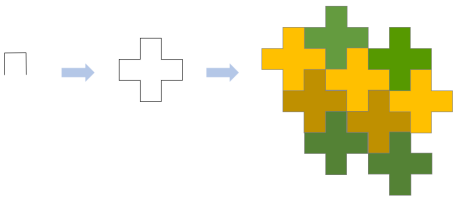


Begriffe

Beim **modularen Entwurf** handelt es sich um ein Prinzip zum Bau von komplexen Systemen aus kleineren Bausteinen (Modulen).

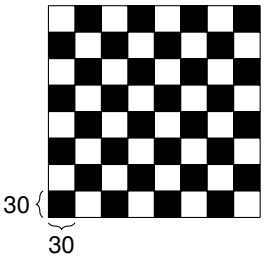
Wir wenden den modularen Entwurf ebenfalls an: Wir schreiben kleine Programmteile, die einfache Aufgaben erfüllen und nennen sie **Module** bzw. Befehle. Durch das Zusammensetzen dieser Module, entsteht ein Programm, das eine schwierigere Aufgabe lösen kann. Die zusammengesetzten Programme können wiederum die Module (Befehle) sein, aus denen noch komplexere Programme aufgebaut sind.

In Beispiel 3.2 haben wir ein Kreuz aus vier kleinen Bausteinen gebaut. Das Kreuz selber kann nun wiederum ein Modul eines noch komplexeren Programms sein, wie die abgebildete Parkettierung:



Aufgabe 3.5

Für ein Online-Schachspiel soll ein Schachbrett programmiert werden mit einer Feldgrösse von 30



Du kannst zum Beispiel folgendermassen vorgehen:

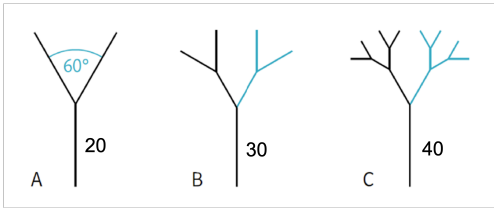
a) Erstelle die Befehle `quadS()` und `quadW()` für ein schwarzes/weisses Quadrat.

b) Verwende die beiden Befehlen in einem neuen Befehl `reihe()` der eine Reihe zeichnet.

c) Verwende `reihe()` um mit dem Befehl `schachbrett()` das ganze Schachbrett zu zeichnen.

Aufgabe 3.6

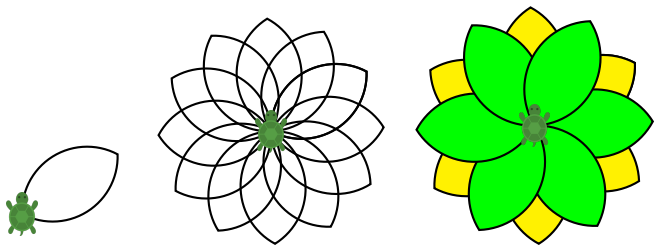
Erstelle einen Befehl `baum1()`, der einen Baum mit Stamm und zwei Ästen mit Länge 20 zeichnet (A). Erstelle einen neuen Befehl `baum2()`, der den Baum erweitert um einen Stamm mit Länge 30 und den Befehl `baum1()` für die Äste verwendet. Entwickle auf gleiche Weise den Befehl `baum3()`.



Aufgabe 3.7

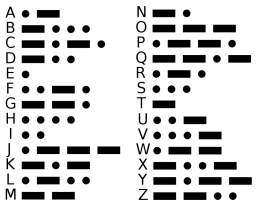
Erstelle einen Befehl `blatt()` aus dem Programmcode für ein Blütenblatt (siehe unten). Verwende den Befehl, um ein Blume zu zeichnen. (die auch gefärbt sein kann)

```
1  from gturtle import*
2  makeTurtle()
3  ht()
4
5  repeat 30:
6      forward(6)
7      rt(4)
8  rt(60)
9  repeat 30:
10     forward(6)
11     rt(4)
```



Akustisches Morsesignal

Mit Morsesignalen können Nachrichten effizient übertragen werden. Das internationale Morsealphabet (Samuel Morse, 1837) kodiert Zeichen mit kurzen und langen Tönen.



Beispiel 3.3

Wir verwenden `playTone(440,t)` um Morsenachrichten zu erzeugen. Das Beispiel zeigt die Umsetzung des Anfangs der Nachricht "WE INTEND".


```
1  from gturtle import*
2
3  playTone(440,200)
4  delay(100)
5  playTone(440,600)
6  delay(100)
7  playTone(440,600)
8  delay(600)
9  playTone(440,200)
10 delay(1200)
11 playTone(440,200)
12 delay(100)
13 playTone(440,200)
```

Aufgabe 3.8

Erstelle Befehle `kurz()` und `lang()` und verwende diese, um wiederum Befehle für die Buchstaben `S()` und `O()` zu definieren. Erzeuge damit die Notnachricht SOS SOS SOS.