1. Zeichnen mit der Python-Turtle

Turtle ist ein Modul in Python, das sich verhält wir ein Zeichenbrett. Es besitzt Funktionen wie `forward(...)` und `left(...)`, welche die Schildkröte umherbewegen. Dabei hinterlässt sie eine Linie als Spur.

a) Als Erstes müssen wir die Programmdatei erzeugen:
- Lege eine neue (noch leere) Datei mit dem IDLE-Editor an (*File* > *New File*)
- Speichere die Datei dann auf der Festplatte und verwende die Dateieindung *.py, also etwa: *meinErstesProgramm.py*

b) Damit das Turtle-Modul benutzt werden kann, muss es importiert werden.
Tippe dazu als erste Zeile Folgendes:

```
from turtle import *
```

c) Nun können alle Befehle aus dem Turtle-Modul verwendet werden.
Die folgenden beiden Befehle veranlassen, dass sich die Schildkröte um 250 Pixel vorwärts bewegt und dann um 60° nach links dreht:

```
forward(250)
left(60)
```

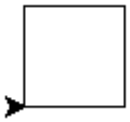
d) Teste nun das Programm (*Run* > *Run Module*).

e) Die Schildkröte wird aktuell als einfacher Pfeil dargestellt. Mit dem Befehl `shape("turtle")` kann jedoch eine richtige Schildkröte daraus gemacht werden.

Das gesamte Programm sieht nun in etwa so aus:

```
from turtle import *
shape("turtle")
forward(250)
left(60)
```

Übung a) Zeichne ein Quadrat so wie im folgenden Bild:



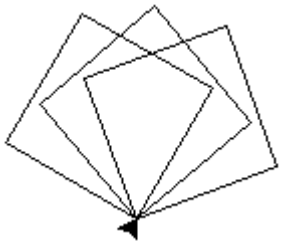
Dafür brauchst du vermutlich einen rechten Winkel, welcher 90˚ beträgt.

Übung b) Zeichne ein Rechteck, ähnlich zu diesem:

<u>Bonus</u>: Zeichne ein Dreieck! In einem gleichseitigen Dreieck (alle Seiten haben die gleiche Länge) hat jede Ecke einen Winkel von 60 Grad.

Übung c) Zeichne drei schräg zur Seite gekippte Quadrate. Experimentiere mit den Winkeln zwischen den einzelnen Quadraten.

Das Bild zeigt 3 Drehungen um jeweils 20 Grad. Probiere auch einmal andere Werte aus!



Übung d) Die geometrische Form kann mit den Funktionen `width(...)` und `color(...)` verändert werden.

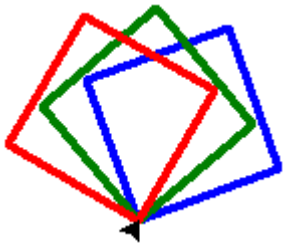
Mit folgendem Befehl werden ab jetzt alle Linien mit der Breite von 5 Pixeln gezeichnet:

```
width(5)
```

Mit folgendem Befehl änderst du die Pinselfarbe zu grün:

```
color("green")
```

Erzeuge folgendes Bild:



2. Arbeit sparen mit Variablen

Wenn du mit den Winkeln und Seitenlängen experimentierst, musst du jedes Mal an verschiedenen Stellen den Code verändern, das ist unpraktisch!

An dieser Stelle kommen Variablen ins Spiel: Du kannst ab nun Python befehlen, jedes Mal wenn du eine bestimmte Variable verwendest, an dieser Stelle etwas anderes einzusetzen. Dieses Konzept ist ähnlich zur Algebra, wo du schreiben könntest: x sei 5. Dann ist x * 2 natürlich 10. In der Syntax von Python hat das die gleiche Bedeutung:

```
x = 5
```

Wenn du nach diesem Befehl `print(x)` ausführst, wird der Wert von x ausgegeben: 5. Du kannst das Gleiche auch mit einem Turtle-Befehl kombinieren:

```
forward(x)
```

Übung a) Sieh dir noch einmal das Programm mit den gekippten Quaddraten aus Übung 1c) an. Lege zwei Variablen an und nenne sie `angle` (für den Winkel) und `length` (für die Seitenlänge). Ersetze nun die Zahlen in den Aufrufen `forward()` und `left()` durch die zwei Variablen.

3. Dinge wiederholen mit Schleifen

Wie du sicher bemerkt hast, enthalten unsere Programme oft Wiederholungen. Programmiersprachen bieten dafür ein mächtiges Werkzeug: Schleifen.

Schleifen sind hilfreich, um mehrere Male etwas zu tun und den Programmcode dafür nur einmal zu schreiben, wie etwa beim Zeichnen der Umrandung einer geometrischen Form.

Hier ist eine Version einer **for-Schleife**. Probiere aus, was passiert:

```
for i in range(10):
    print(i)
```

Die `range(n)`-Funktion kann man als eine Abkürzung für die Werte `0, 1, 2, ..., n-1` sehen, die der Reihe nach durchlaufen werden.

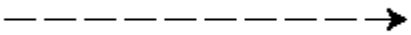
Du kannst auch Elemente deiner Wahl in einer Schleife verarbeiten:

```
total = 0
for i in 5, 7, 11, 13:
    print(i)
    total = total + i
print(total)
```

Schreib das Beispiel ab und und überprüfe, ob es so funktioniert wie du dir vorstellst.

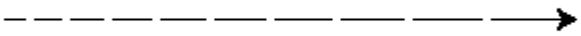
Wichtig: Alle Zeilen, die wiederholt werden sollen, müssen eingerückt werden. Dadurch weiß Python, welche Zeilen zur for-Schleife gehören und welche danach kommen.

Übung a) Zeichne eine gestrichelte Linie, mit Hilfe einer Schleife:



Mit `penup()` erreichst du, dass die Schildkröte beim Bewegen nicht zeichnet, mit `pendown()` hinterlässt sie bei den folgenden Bewegungen wieder eine Linie.

Übung b) Lasse die Striche immer länger werden:



Übung c) Die Quadrate aus 1.) erforderten viele wiederholte Textzeilen. Schreibe ein Programm, das ein Quadrat wie in Übung 1a) malt, aber mit weniger Codezeilen.
Tipp: Verwende eine Schleife, die vier Mal durchlaufen wird.

Übung d) Verschachtele 2 Schleifen, um die Quadrate aus Übung 1c) mehrmals zu zeichnen.

4. Befehle selbst erfinden mit Funktionen

Mit Funktionen hat man die Möglichkeit, mehrere Anweisungen in einer einzigen neuen Anweisung zusammenzufassen.

In Python kann eine Funktion mit dem Schlüsselwort `def` definiert werden:

```
def line_without_moving():
    forward(50)
    backward(50)
```

Diese von uns definierte Funktion heißt `line_without_moving` und ist praktisch eine Abkürzung für zwei Bewegungen: ein Schritt vorwärts und ein Schritt rückwärts.

Um sie zu benutzen ***(aufzurufen)***, schreibe ihren Namen gefolgt von runden Klammern:

```
line_without_moving()
right(90)
line_without_moving()
right(90)
line_without_moving()
```

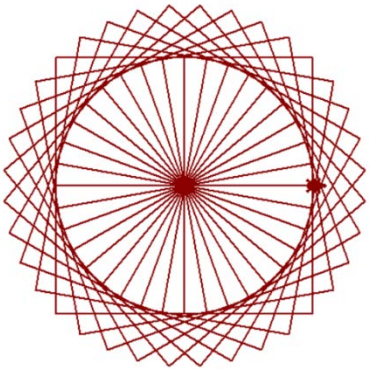
Funktionen können auch innerhalb von anderen Funktionen verwendet werden. Eine Funktion, die rechts abgebildete sternförmige Figur zeichnet, könnte beispielsweise so definiert und aufgerufen werden:

```
def star():
    for i in range(5):
        line_without_moving()
        right(360 / 5)

star()
```

<u>Wichtig</u>: Beachte auch hier die korrekte Einrückung, damit Python erkennt, welche Zeilen zur Funktion und welche zur `for`-Schleife innerhalb der Funktion gehören.

Übung a) Schreibe eine Funktion, die ein Quadrat zeichnet. Versuche anschließend, die Funktion zu verwenden, um das Programm mit den gekippten Quadraten aus Übung 3d) zu verbessern. Male aus Quadraten eine ganze Rosette:



5. Noch flexibler: Funktionen mit Parametern

Unsere bisherigen Funktionen machen jedes Mal genau dasselbe: Die Quadrat-Funktion aus Übung 4a) kann beispielsweise nur Quadrate einer Größe zeichnen. Um dies zu ändern, benötigen wir *Parameter*. So können die Werte innerhalb der Funktion bei jedem Aufruf anders sein.

Zum Beispiel können wir die Funktion `line_without_moving()` verbessern, indem wir ihr einen Parameter übergeben:

```
def line_without_moving(length):
    forward(length)
    backward(length)
```

Der Parameter ist eine Variable, die nur innerhalb der Funktion bekannt ist. Wir verwenden die Funktion, indem wir sie mit dem aktuellen Wert für den Parameter aufrufen:

```
line_without_moving(30)
right(90)
line_without_moving(40)
```

Dadurch zeichnet sie nicht jedes Mal eine Linie der Länge 50, sondern jeweils eine Linie der Länge, die wir als Parameter angegeben haben (in obigem Beispiel also 30 und 40).

Wir haben von Anfang an bereits Funktionen mit Parametern verwendet, wie z.B. `forward(100)` oder `left(90)`.

Für eine Funktion können beliebig viele Parameter definiert werden. Die einzelnen Parameter sind dabei durch Kommas getrennt und haben alle unterschiedliche Namen:

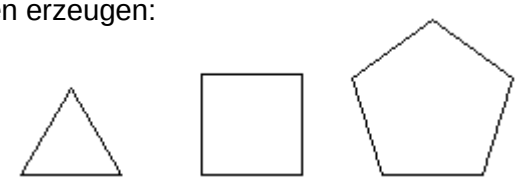
```
def line_without_moving(length, angle):
    left(angle)
    forward(length)
    backward(length)
```

Übung a) Schreibe eine Funktion, die ein Quadrat beliebiger Größe zeichnet.

Übung b) Schreibe eine Funktion, die eine Form mit beliebiger Größe und beliebiger Anzahl von Seiten zeichnet:

```
def shape(length, edges):
    ...
```

Aufrufe von `shape(50,3)`, `shape(50,4)`, `shape(50,5)` sollten beispielsweise folgende Formen erzeugen:



Tipp: Die Summe der Außenwinkel in jeder Form ist immer 360 Grad!

6. Bedingungen stellen mit Verzweigungen

Verzweigungen bieten die Möglichkeit, bestimme Programmabschnitte nur dann auszuführen, wenn eine Bedingung gilt.

Konzeptionell wird eine Verzweigung folgendermaßen formuliert:

```
if Bedingung:
    # Code, der ausgeführt wird, wenn die Bedingung wahr ist
else:
    # Code, der ausgeführt wird, wenn die Bedingung falsch ist
```

Die *Bedingung* ist dabei ein Ausdruck, der zu `True` oder `False`, also *wahr* oder *falsch* ausgewertet wird. Zum Beispiel kann man testen, ob eine Variable einen bestimmten Wert hat (ist gleich: `==`), oder ob eine Zahl größer als (`>`) oder kleiner als gleich (`<=`) einer anderen Zahl ist.

Probiere einemal das folgende Beispiels aus und sieh dir an, wie sich das Programm verhält, wenn du für `zahl` Werte größer oder kleiner als `50` wählst:

```
zahl = 75
if zahl > 50:
    print("Die Zahl ist größer als 50")
else:
    print("Die Zahl ist kleiner gleich 50")
```

Der `else`-Block kann bei einer if-Abfrage weggelassen werden. Falls du ihn auslässt und die Bedingung als `False` ausgewertet wird, passiert einfach gar nichts.

Mit der `input()`-Funktion kann der Benutzer aufgefordert werden, einen Text einzutippen. Dies können wir verwenden, um die Schildkröte zu steuern. Um es nicht zu kompliziert zu machen, erlauben wir nur die zwei Befehle „links“ und „rechts“.

Schreibe folgende Funktion ab und rufe sie mindestens einmal mit `move()` auf:

```
def move():
    direction = input("links oder rechts?")
    if direction == "links":
        left(60)
    if direction == "rechts":
        right(60)

    turtle.forward(50)
```

Bei jedem Aufruf von `move()` wirst du aufgefordert, entweder „links“ oder „rechts“ einzutippen. Was passiert, wenn du etwas ganz anderes eintippst?

Übung a) Benutze if-Abfragen, um in dem Beispiel mit den farbigen gekippten Quadraten aus Übung 2d) die Farbwahl zu steuern.