

Happy little turtles.

## 4. Turtle-Grundlagen

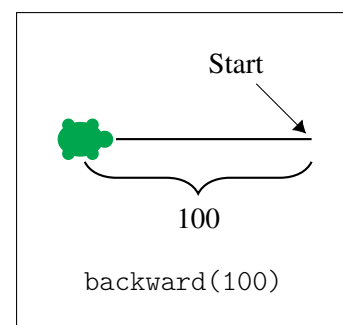
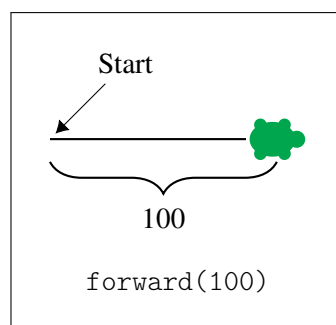
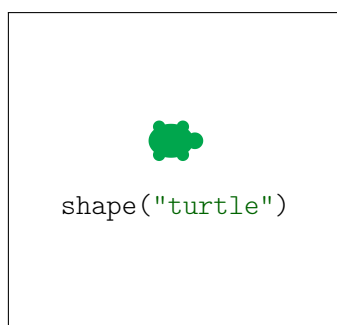
Sie kennen und verwenden nach diesem Kapitel die Python-Befehle für die folgenden Aktionen:

- ☐ Die Turtle eine Anzahl Schritte vorwärts bzw. rückwärts bewegen.
- ☐ Die Turtle um einen beliebigen Winkel (in Grad) nach rechts bzw. links drehen.
- ☐ Die Turtle in den „Wandermodus“ bzw. „Zeichenmodus“ versetzen.
- ☐ Das Icon (kleines, ausgefülltes Dreieck) anpassen.

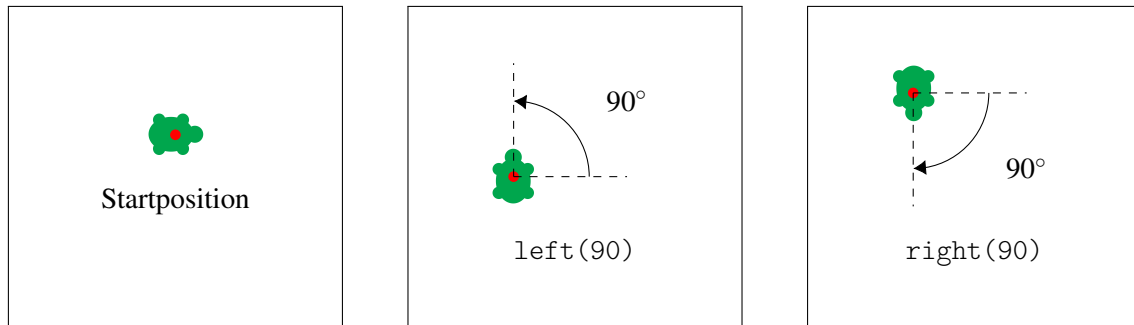
### 4.1 Erste Schritte mit der Turtle

Wir besprechen nun die grundlegenden Konzepte.

**H** Die Turtle befindet sich zu Beginn in der Mitte des Fensters und blickt nach **rechts**. Die Turtle befindet sich zu Beginn stets im **Zeichenmodus**. Ist die Turtle im Zeichenmodus, dann besitzt die Schildkröte einen Stift und zeichnet Ihren Weg auf. Dies geschieht nur, wenn die Turtle sich vorwärts oder rückwärts bewegt.



Mit `shape("turtle")` können wir das **Icon** in eine Turtle ändern. Dies hat keinen Einfluss auf die Bewegungsabläufe der Turtle. Es wird lediglich das Icon ausgetauscht. Das Icon ändert sich nur dann, wenn wir es in das Programm einbauen. Das Icon verändert sich erst nachdem die entsprechende Code-Zeile verarbeitet wurde. Mit dem Befehl `forward(distance)` bewegt sich die Turtle um `distance` Schritte vorwärts. Sie läuft dabei in **Blickrichtung**. Der Befehl `back(distance)` bewirkt das Gegenteil. Die Turtle bewegt sich um `distance` Schritte rückwärts. Sie läuft **entgegen** der **Blickrichtung**. Die Anzahl der „Schritte“ können wir frei wählen. Mit `right(angle)` können wir die Turtle um `angle` Grad nach **rechts** drehen. Den Befehl `left(angle)` können wir gleichermassen für eine Linksdrehung verwenden. Die Turtle dreht



sich dabei um `angle` Grad nach **links**. Das Drehen der Turtle bezieht sich **immer** auf die aktuelle Blickrichtung der Turtle<sup>1</sup>. Den Winkel können wir frei wählen. Die Drehung bewegt die Turtle weder nach vorn noch zurück - es wird **nichts** gezeichnet. Prinzipiell können die Befehle zur Fortbewegung und Drehung beliebig oft und mit unterschiedlichen Zahlen in einem Programm verwendet werden.

■ **Beispiel 4.1** Das Listing 4.1 zeigt, wie wir die Turtle mit mehreren Befehlen mehrmals bewegen und drehen können. In Abbildung 4.1 wird das Resultat gezeigt.

```
1 import turtle
2
3 turtle.shape("turtle")
4 turtle.left(90)
5 turtle.forward(50)
6 turtle.right(90)
7 turtle.forward(100)
8 turtle.right(135)
9 turtle.forward(100)
10 turtle.done()
```

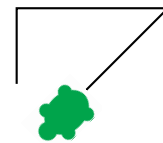


Abbildung 4.1: Resultat, wenn wir das Programm aus Listing 4.1 ausführen.

Listing 4.1: Beispielprogramm, welches die Figur aus Abbildung 4.1 zeichnet (`bsp_1.py`).

■

## 4.2 Programmausführung und Clean Code


Wir klären zunächst die Art und Weise, wie ein Python-Programm durch den Computer ausgeführt wird und gehen dann auf den Programmierstil ein.

**Definition 5 — Programmausführung.** Ein Python-Programm wird von **oben** nach **unten** ausgeführt. Der Computer liest Zeile für Zeile des Programms und führt nacheinander die entsprechenden Befehle aus. Bei einer Leerzeile passiert „nichts“. Es gibt Befehle, die Code-Zeilen überspringen oder zu einer vorherigen Code-Zeile zurückgehen.

Python führt somit ein Programm stets ab Zeile 1 aus. Damit Programme übersichtlich und leserlich bleiben, werden bei der Software-Entwicklung gewisse Regeln vereinbart. Typischerweise halten sich **alle** in einem Team an diese Regeln. Wenn wir gegenseitig Code austauschen und betrachten, findet sich somit jeder gleich zurecht. Ausserdem sollen die Regeln aufzeigen, wie wir unseren Programmierstil verbessern und dadurch verständlichere Programme erzeugen können. Wir fassen diese Regeln unter dem Begriff **Clean Code**<sup>2</sup> zusammen.

<sup>1</sup>Manchmal hilft folgender „Trick“: Versetzen Sie sich in die Turtle und führen Sie dann die Drehung aus.

<sup>2</sup>Wikipedia fasst den Begriff „perfekt“ zusammen: [https://de.wikipedia.org/wiki/Clean\\_Code](https://de.wikipedia.org/wiki/Clean_Code)

 **Clean Code 1 — Ein Befehl pro Zeile.** Wir notieren pro Zeile **einen** Befehl. Alle **import**-Befehle werden in den ersten Zeilen der Python-Datei notiert. Nach dem letzten **import**-Befehl fügen wir eine **Leerzeile** ein.

### 4.3 Wandermodus

Mit `penup()` bringen wir die Turtle in den **Wandermodus**. Im Wandermodus bewegt sich die Turtle, **ohne** mit dem Stift zu zeichnen. Der Befehl `pendown()` bringt die Turtle in den **Zeichenmodus**. Im Zeichenmodus hat die Schildkröte einen Stift und zeichnet.

■ **Beispiel 4.2** Listing 4.2 zeigt, wie wir den Wander- und Zeichenmodus verwenden.

```
1 import turtle
2
3 turtle.shape("turtle")
4 turtle.forward(50)
5 turtle.penup()
6 turtle.forward(50)
7 turtle.pendown()
8 turtle.forward(50)
9 turtle.done()
```

Listing 4.2: Beispielprogramm, welches die Figur aus Abbildung 4.2 zeichnet (`bsp_2.py`).



Abbildung 4.2: Resultat, wenn wir das Programm aus Listing 4.2 ausführen.

■

### 4.4 Programmierauftrag

Lesen Sie die rote Box und erfüllen Sie den Auftrag.

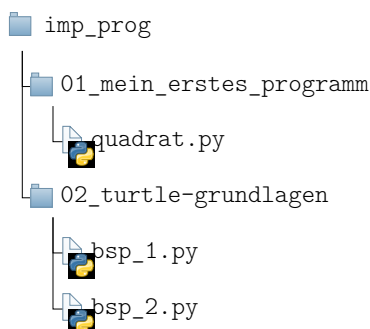


Abbildung 4.3: Ordnerstruktur in der IDE.

**Wichtig!** Tippen Sie den Code der beiden Listings aus diesem Kapitel ab und führen Sie das Programm aus. Erstellen Sie dazu einen neuen Ordner und zwei neue Python-Dateien. Die Datei- und Ordnerstruktur sollte nun wie in Abbildung 4.3 aussehen.

■