

```
def getSolutionCosts(navigationCode):
    fuelStopCost = 15
    extraComputationCost = 8
    thisAlgorithmBecomingSkynetCost = 999999999
    waterCrossingCost = 45
```

GENETIC ALGORITHMS TIP:
ALWAYS INCLUDE THIS IN YOUR FITNESS FUNCTION

Just make sure you don't have it maximize instead of minimize.

8. Funktionen

Mit eigenen Funktionen machen wir unsere Programme übersichtlicher und verständlicher. Wir können damit auch auf Tastatur- und Mausereignisse reagieren. Die Lernziele sind:

- ☐ Sie erklären, was eine Funktion ist und warum die Verwendung nützlich sein kann.
- ☐ Sie geben Beispiele für bestehende Funktionen.
- ☐ Sie definieren eine Funktion in Python und rufen diese an geeigneter Stelle auf.
- ☐ Sie verwenden Funktionen, um Code bei einem Tastaturereignis auszuführen.

8.1 Zwei Quadrate ☺

Wir können mit dem Programm aus Listing 22 die Figur aus Abbildung 11 zeichnen.

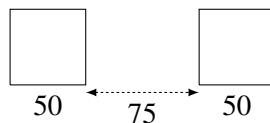


Abbildung 11: Zwei Quadrate mit Abstand 75.

```
1 import turtle as t
2
3
4 def quadrat_50():
5     for _ in range(4):
6         t.fd(50)
7         t.lt(90)
8
9
10 quadrat_50()
11 t.pu()
12 t.fd(125)
13 t.pd()
14 quadrat_50()
15
16 t.done()
17
```

Listing 22: `zwei_quadrate.py`

Wir gestalten den Code übersichtlich und sparen Tipparbeit, indem wir eine Funktion definieren und zweimal aufrufen.

Definition 12 — Funktion. Mit einer Funktion fassen wir Quellcode unter einem **Namen** zusammen und verwenden ihn an mehreren Stellen wieder, ohne den Code kopieren zu müssen.

Listing 22 zeigt die Verwendung einer Funktion am Beispiel der beiden Quadrate. Bisher haben wir „nur“ bestehende Funktionen aufgerufen. Nun definieren wir zunächst eigene Funktionen und rufen diese dann auf. Abbildung 12 zeigt diesen Zusammenhang.

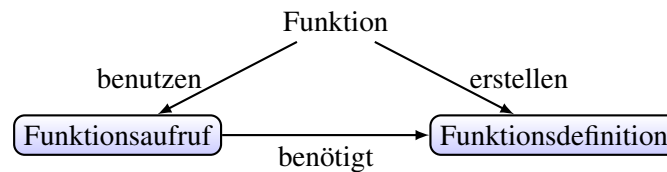


Abbildung 12: Aufbau von Funktionen

Wichtig! — **Funktionsdefinition** \neq **Funktionsaufruf**. Grundsätzlich gilt: Bevor eine Funktion aufgerufen werden kann, muss sie definiert werden. Wenn man eine Funktion definiert, dann werden die Befehle der Funktion noch **nicht** automatisch ausgeführt. ■

8.2 Funktionsdefinition

Eine Funktionsdefinition besteht aus einem **Funktionskopf** und einem **Funktionskörper**. Zuerst wird der Funktionskopf notiert, dann der Funktionskörper. Listing 23 zeigt den allgemeinen Aufbau einer Funktionsdefinition und Listing 24 ein konkretes Beispiel einer Funktionsdefinition.

```

1
2
3 def FUNKTIONSNAME():
4     BEFEHL_1
5     BEFEHL_2
6     ...
7
8
  
```

Listing 23: Allgemeine Struktur

```

1
2
3 def quadrat_50():
4     for _ in range(4):
5         t.fd(50)
6         t.lt(90)
7
8
  
```

Listing 24: Beispiel einer Funktionsdefinition

8.2.1 Funktionskopf

Der Funktionskopf beginnt **immer** mit dem Schlüsselwort¹ **def** (eine Abkürzung für **definition**). Das Schlüsselwort **def** bedeutet, dass nun die Definition der Funktion beginnt. Der Funktionskopf **muss** mit einem Doppelpunkt **:** abgeschlossen werden.

8.2.2 Funktionskörper

Nur die **engerückten** Befehle gehören zum Funktionskörper und damit zur Funktionsdefinition. Diese Befehle werden beim Aufruf der Funktion ausgeführt. Innerhalb des Funktionskörpers können beliebige Befehle notiert werden.

 **Clean Code** — **Leerzeilen bei der Funktionsdefinition.** Vor **und** nach einer Funktionsdefinition sind **zwei Leerzeilen** einzufügen.

 **Clean Code** — **Einrückung 2.** Die Einrückung (engl. indentation) des **Funktionskörpers** erfolgt mit der Tabulatortaste.

¹Wir haben bereits weitere Schlüsselwörter kennengelernt: **import**, **for** und **in**.

8.3 Funktionsaufruf

Mit einem **Funktionsaufruf** können wir eine **Funktion ausführen**. Ein Funktionsaufruf erfolgt durch die Notation des **Funktionsnamens** und der **Klammern**. Den Funktionsaufruf kennen wir schon. Wir haben schon immer Funktionen aufgerufen, die *andere* Programmierer geschrieben haben. Neu ist nun, dass wir auch noch unsere eigenen Funktionen aufrufen.

■ **Beispiel 21** Ausgewählte Funktionsaufrufe bereits vorhandener Funktionsdefinitionen:

- `forward(100)`
- `left(90)`
- `pencolor("red")`
- `randrange(50, 101)`
- `choice(["red", "green", "blue"])`
- `sqrt(42)`

H Der Funktionsaufruf eigener Funktionen unterscheidet sich nicht vom bisherigen Funktionsaufruf. Wir verwenden **weder** `def` beim Funktionsaufruf noch den Doppelpunkt. Einziger Unterschied: Unsere Funktionsdefinitionen haben noch keine Parameter für den Funktionsaufruf mit einem Argument. Aber auch das können wir noch hinzufügen.

Wir können eine Funktion beliebig oft aufrufen. Egal wann wir eine Funktion aufrufen, die Befehle innerhalb des Funktionskörpers werden immer genau einmal pro Funktionsaufruf ausgeführt.

Wichtig! Wenn sich die Funktionsdefinition in derselben Python-Datei wie der Funktionsaufruf befindet, ist keine `import`-Anweisung erforderlich. Die Funktion kann direkt aufgerufen werden. Andernfalls ist die übliche Notation mit dem „Punkt“ zu verwenden.

■ **Beispiel 22 — Zwei Quadrate.** In Listing 22 rufen wir zweimal unsere eigene Funktion auf. In Zeile 10 und 14: `quadrat_50`.

8.4 Funktionen kombinieren

Natürlich können wir auch mehrere Funktionen definieren und die Funktionsaufrufe miteinander kombinieren. Listing 25 zeigt eine Erweiterung des Beispiels aus Listing 22.

```


1  import turtle as t
2
3
4  def quadrat_50():
5      for _ in range(4):
6          t.fd(50)
7          t.lt(90)
8
9
10 def quadrate():
11     for _ in range(4):
12         quadrat_50()
13         t.pu()
14         t.fd(175)
15         t.lt(90)
16         t.pd()
17
18
19 quadrate()
20 t.done()
21

```

Listing 25: Zwei Funktionsdefinitionen. In der zweiten Funktionsdefinition wird die erste Funktion durch die Schleife mehrfach aufgerufen.

8.5 Wie wählen wir Funktionsnamen?

Im Rahmen der üblichen Regeln sind wir bei der Wahl des Funktionsnamens grundsätzlich frei. Allerdings sollten wir den Funktionsnamen so wählen, dass allein aus dem Funktionsnamen ersichtlich ist, was die Befehle im Funktionskörper bewirken.

 **Clean Code** — **Sinnvolle Funktionsnamen.** Wir wählen **Funktionsnamen** so, dass wir sofort verstehen, was die Funktion macht. Wir verwenden keine Grossbuchstaben und trennen einzelne Wörter durch einen Unterstrich (_). Dadurch machen wir den Code **lesbarer**.

■ **Beispiel 23 — Zwei Quadrate.** Der Funktionsname `quadrat_50` aus Listing 22 deutet direkt darauf hin, was die Funktion macht: Es wird ein Quadrat mit der Seitenlänge 50 gezeichnet.

8.6 Tastaturereignisse verarbeiten

Mit der Funktion `onkeypress` registrieren wir eine **Funktion**, die beim Drücken der entsprechenden Taste einmalig ausgeführt wird. Wird die Taste zweimal hintereinander gedrückt, wird die Funktion zweimal hintereinander ausgeführt. Das **erste Argument** ist der **Funktionsname**, das **zweite Argument** ist die **Tastenbezeichnung** als **String**. Damit die Tastaturereignisse verarbeitet werden, müssen wir **nach der Registrierung einmal** die `listen`-Funktion aufrufen.

```

1  import turtle as t
2  import random as r
3
4
5  def quadrat_50():
6      for _ in range(4):
7          t.fd(50)
8          t.lt(90)
9
10
11 def move_forward_randomly():
12     t.pu()
13     laenge = r.randrange(100, 201)
14     t.fd(laenge)
15     t.pd()
16
17
18 def rotate_left_randomly():
19     t.pu()
20     winkel = r.randrange(0, 361)
21     t.lt(winkel)
22     t.pd()
23
24
25 t.onkeypress(quadrat_50, "q")
26 t.onkeypress(move_forward_randomly, "f")
27 t.onkeypress(rotate_left_randomly, "l")
28 t.listen()
29 t.done()
30

```

Listing 26: Es werden drei Funktionen registriert. Wenn wir zum Beispiel auf die Taste „f“ drücken, dann wird die Funktion `move_forward_randomly` ausgeführt.