

Anweisungen, Funktionen und Parameter (Beispiel Turtle)

Das Zeichnen vom Haus des Nikolaus kann als informatisches Problem aufgefasst werden. Zur Lösung eines solchen **Problems** wird ein **Algorithmus** (Verfahren) eingesetzt. Meist gibt es nicht nur einen Algorithmus zur Lösung eines Problems in der Informatik.

Beispiel einer Problemlösung am Beispiel des Haus vom Nikolaus (mit Absetzen!)

Algorithmus 1

1. Zeichne Quadrat
2. Zeichne rechtwinkliges Dreieck (auf dem Quadrat)
3. Zeichne Kreuz (im Quadrat)

Alternative zu obigem Vorgehen

Algorithmus 2

1. Zeichne Quadrat mit innenliegenden Diagonalen
2. Zeichne Dreieck

Das „Problem“ aus 1. des o.a. Algorithmus 2 lässt sich in zwei **Teilprobleme** aufteilen. Lösen wir die beiden neuen Teilprobleme, so haben wir auch das Problem aus 1. gelöst!

1. "Zeichne Quadrat mit innenliegenden Diagonalen" - wird jetzt gelöst durch zwei neue Teilprobleme:
 - Zeichne Quadrat (erstes neues Teilproblem)
 - Zeichne Diagonalen des Quadrats (zweites neues Teilproblem)
2. Zeichne rechtwinkliges Dreieck

Modularisierung in der Informatik - (Teil)-Probleme mit Hilfe von Funktionen lösen

(Skript „August 2017“ Seiten 14-19, Kapitel Parameter, Repetitionen, Modularität)

In der Informatik besteht ein fundamentales **Modularisierungsprinzip**, das ein Denken und Arbeiten in autonomen Bausteinen vorsieht.

Bei der Implementierung des Algorithmus in eine Programmiersprache sind sogenannte **Funktionen** ein Teil dieser autonomen Bausteine zur Lösung eines (informatischen) Problems bzw. der entstandenen Teilprobleme. Fast alle Programmiersprachen verfügen über Funktionen. Jede Funktion hat einen Namen und gegebenenfalls einen oder mehrere **Parameter**. Manche Funktionen liefern auch einen **Rückgabewert** (zurück). In Funktionen werden Anweisungen und Kontrollstrukturen und auch andere Funktionen verwendet. Eine Funktion, die in ihrer Funktionsdefinition sich selbst verwendet (sich selbst wieder aufruft), wird **rekursive Funktion** genannt.

Funktionen werden in Python im Allgemeinen an **den Anfang des (Haupt-)Programms** nach dem Import von Modulen (Bibliotheken) definiert.

Danach kommt erst der Programmtext mit dem Aufruf der Funktionen.

Beispiel

1. Definition einer Funktion

```
def zeichne_quadrat(): # Funktionsname zeichne_quadrat, kein Parameter
    forward (100) # Einrückungen beachten!
    right(90)
    forward (100)
    right(90)
    forward (100)
    right(90)
    forward (100)
    # right(90) optional, dann steht die Turtle wieder in der gleichen Richtung
    wie vor dem Aufruf der Funktion!!
# end of function
```

2. Aufruf der Funktion `zeichne_quadrat`:

```
...
setPenColor("red")
zeichne_quadrat() # dieser Funktionsaufruf ist auch eine Anweisung!
penUp()
...
```

Aufgabe

- Definiere eine Funktion `zeichne_quadrat_mit_diagonalen()` und eine Funktion `zeichne_rechtwinkliges_dreieck()`
- Rufe beide Funktionen nacheinander auf. Was fällt auf? Wird ein Haus des Nikolaus gezeichnet? Wenn nein, überprüfe die Start- und Endbedingungen der Funktionen: wo muss die Turtle vor und nach Aufruf der Funktionen stehen? Wo lassen sich diese Vorbedingungen einbauen?
 - Überprüfe somit die **Eingangsvoraussetzungen** für den Aufruf der Funktion:
 - In welche Richtung muss die Turtle zeigen?
 - Muss der Stift abgesenkt sein (malbereit?)
 - Überprüfe die **Ausgangsvoraussetzungen** deiner Funktionen:
 - wie steht die Turtle nach Aufruf
 - Ist der Stift malbereit?
- Das fertige (Haupt-)Programm soll analog zu Algorithmus 2 (s.o) nur aus folgenden beiden Funktionsaufrufen bestehen:

```
zeichne_quadrat_mit_diagonalen()
zeichne_rechtwinkliges_dreieck()
```

- **Kommentiere** den Funktionskopf deiner Funktionen entsprechend den Start- und Endbedingungen.

Problem: wir wollen Quadrate verschiedener Größen haben. Wir brauchen eine Funktion, die die Länge einer Quadratseite bekommt, sodass wir beim Aufruf der Funktion nur als Wert eine Größe übergeben brauchen.

Lösung: wir benutzen Parameter und ändern unsere Funktion `zeichne_quadrat` entsprechend ab

1. Definition dieser Funktion

```
def zeichne_quadrat(seitenlaenge):
    forward (seitenlaenge)
    right (90)
    forward (seitenlaenge)
    right (90)
    forward (seitenlaenge)
    right (90)
    forward (seitenlaenge)
    right (90)
```

2. Aufruf dieser Funktion:

```
...
setPenColor("red")
zeichne_quadrat(218) # ein Quadrat mit der Seitenlaenge 218 wird gezeichnet
zeichne_quadrat(333) # ein Quadrat mit der Seitenlaenge 333 wird gezeichnet
penUp()
```

...

Problem: wir wollen mehrere gleiche wiederholende Anweisungen der obigen Funktion zusammen fassen.

Lösung: Dabei hilft uns die Kontrollstruktur Schleife. In diesem Fall benutzen wir eine repeat-Schleife (Kapitel 2.6, Seiten 16ff.) und passen unsere Funktion *zeichne_quadrat* entsprechend an:

Hinweis: beachte, dass die Benennung des Parameters in der Definition frei gewählt werden kann!

```
def zeichne_quadrat (seite):
    repeat 4: # ACHTUNG Einrückung!!
        forward(seite)
        right(90)

    left(90) # gehört nicht mehr zur repeat-Schleife, aber immer noch zur
Funktionsdefinition!
right (90) # gehört nicht mehr zur Funktionsdefinition!
```