Anweisungen, Funktionen und Parameter (Beispiel Turtle)

Das Zeichnen vom Haus des Nikolaus kann **als informatisches Problem** aufgefasst werden. Zur Lösung entwickeln wir einen **Algorithmus** (Verfahren), also eine eindeutige Folge von Anweisungen.
Meist gibt es nicht nur einen Algorithmus zur Lösung eines Problems in der Informatik.

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

Eine Möglichkeit stellt der **1. Algorithmus** dar:
1.1. Zeichne Quadrat
1.2. Zeichne rechtwinkliges Dreieck (auf dem Quadrat)
1.3. Zeichne Kreuz/Diagonalen (im Quadrat)

Eine weitere alternative Möglichkeit, die wir uns jetzt näher anschauen wollen, zeigt uns der
2. Algorithmus:
2.1. Zeichne Quadrat mit innenliegenden Diagonalen
2.2. Zeichne Dreieck

Das Teilproblem 2.1 des o.a. **2. Algorithmus** lässt sich wiederum in zwei kleinere Teilprobleme zerlegen. Wenn wir beide lösen und ihre Lösungen **geeignet miteinander verbinden**, erhalten wir eine Lösung für das ursprüngliche Teilproblem!
2.1. "Zeichne Quadrat mit innenliegenden Diagonalen" - wird jetzt gelöst durch zwei **neue Teilprobleme:**

- Zeichne Quadrat (erstes neues Teilproblem)
- Zeichne Diagonalen des Quadrats (zweites neues Teilproblem)

2.2. Zeichne rechtwinkliges Dreieck (wie gehabt)

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

(Skript „August 2017“ Seiten 14-19, Kapitel Parameter, Repetitionen, Modularität)
Ein grundlegendes Prinzip der Informatik ist die **Modularisierung**. Dabei wird ein komplexes Problem in kleinere, möglichst eigenständige Teilprobleme zerlegt. Diese werden durch einzelne Bausteine, beispielsweise **Funktionen**, gelöst. Damit die Bausteine zusammenarbeiten können, müssen ihre **Schnittstellen** aufeinander abgestimmt sein. Bei einer Turtle-Funktion gehören dazu insbesondere die benötigten Parameter sowie die Bedingungen, die vor und nach ihrer Ausführung gelten.
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**.
Eine Funktion fasst mehrere **Anweisungen** unter einem Namen zusammen. Sie kann beim Programmablauf beliebig oft aufgerufen werden.
Funktionen können **Parameter** besitzen, über die ihnen beim Aufruf **Werte** übergeben werden. Manche Funktionen liefern auch einen **Rückgabewert** (zurück). In Funktionen werden Anweisungen und Kontrollstrukturen und auch andere Funktionen verwendet. Innerhalb einer Funktion können auch andere Funktionen aufgerufen werden. So eine Funktion, die in ihrer Funktionsdefinition sich selbst verwendet (sich selbst wieder aufruft), wird **rekursive Funktion** genannt.
Wir vereinbaren: Funktionen werden an **den Anfang des (Haupt-)Programms** nach dem Import von Modulen (Bibliotheken) definiert. Erst danach kommt 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) 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(= neuer Befehl)!
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? Welche Bedingungen müssen vor dem Funktionsaufruf erfüllt sein, und welche Zustandsänderungen sollte die Funktion selbst übernehmen?
  Untersuche für jede Funktion:
  - **Startbedingungen**: An welcher Position muss sich die Turtle befinden? In welche Richtung muss sie zeigen? Muss der Stift abgesenkt sein?
  - **Endbedingungen**: An welcher Position befindet sich die Turtle nach dem Funktionsaufruf? In welche Richtung zeigt sie? Ist der Stift abgesenkt?
- Das fertige (Haupt-)Programm soll analog zu Algorithmus 2 (s.o) nur aus den folgenden beiden Funktionsaufrufen bestehen:
  ```
  zeichne_quadrat_mit_diagonalen()
  zeichne_rechtwinkliges_dreieck()
  ```
- ***Kommentiere** den Funktionskopf deiner Funktionen mit 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()
…
```

seitenlaenge ist der **Parameter** der Funktion. Beim Funktionsaufruf wird mit 218 ein konkreter **Wert** übergeben. Diesen Wert bezeichnet man als **Argument**. Innerhalb der Funktion kann anschließend mit dem Parameter gearbeitet werden**.**
Problem: wir wollen mehrere gleiche wiederholende Anweisungen der obigen Funktion zusammenfassen.
Lösung: Dabei hilft uns die Kontrollstruktur Schleife. In diesem Fall benutzen wir eine repeat-Schleife (Kapitel 2.5, 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 Funktionsdefini-
tion!
right (90) # gehört nicht mehr zur Funktionsdefinition!
```