

Kapitel 4: Algorithmische Grundstrukturen und ihre Darstellung

Bisher haben wir ausschließlich Programme gesehen, bei denen der Programmcode Zeile für Zeile nacheinander ausgewertet wird. Mit Hilfe **algorithmischer Grundstrukturen** können wir von diesem Verhalten abweichen.


Algorithmische Grundstrukturen steuern dabei den Ablauf eines Programms. Mit ihnen könnt ihr Anweisungsblöcke definieren, die nur unter einer bestimmten Bedingung ausgeführt werden, oder auch solche, deren Ausführung mehrfach erfolgt. Hierfür unterscheiden wir **Sequenzen**, **Verzweigungen** und **Schleifen**, welche wir uns - inklusiv dreier **Darstellungsformen** - nachfolgend genauer anschauen.

4.1. Darstellungsformen

Programme und Algorithmen können auf verschiedenste Art und Weise dargestellt werden. Zu nennen sind hierbei die Darstellungsformen **Pseudocode**, **Code** und **Struktogramm**.

Der **Pseudocode** ist eine natürlichsprachliche Formulierung des Programms. Er versucht in einfacher und nachvollziehbarer Sprache die Struktur eines Programms wiederzugeben. Ihr beschreibt also umgangssprachlich was passieren soll, zum Beispiel "eine Zahl eingeben" oder "das Ergebnis ausgeben".

Der **Code** ist natürlich abhängig von der gewählten Programmiersprache. Wir werden hierbei ausschließlich Python-Code nutzen.

Das **Struktogramm (auch: Nassi-Shneiderman-Diagramm)** versucht mit Hilfe einfacher geometrischer Formen den Programmablauf zu visualisieren und in einer strukturierten Art und Weise darzustellen. Ein digitales Tool zur Erstellung von Struktogrammen steht unter  [Struktog](#). für euch bereit.

Diese Darstellungsformen werden nachfolgend - bei den algorithmischen Grundstrukturen - mit Beispielen untermauert.

Zu den Struktogrammen sei erwähnt, dass die verwendeten Farben **nicht** vorgeschrieben, sondern lediglich Ergebnis des verwendeten Tools sind.

4.2. Sequenz

Bei der **Sequenz** handelt es sich schlichtweg um eine **Abfolge von Anweisungen**.

Diese werden dabei nacheinander aufgeschrieben und von oben nach unten - auch als **iterativ** bezeichnet - abgearbeitet. Alle bisher verwendeten Codeschnipsel waren demnach Sequenzen.

Visualisieren wir dies an einem einfachen **Beispiel**: Dabei wird eine Zahl eingegeben, das Quadrat dieser Zahl berechnet und die berechnete Quadratzahl ausgegeben.

```
In [ ]: eingabe = input("Gib eine Zahl ein: ")
        zahl = int(eingabe)
        quadrat_zahl = zahl**2
        print(quadrat_zahl)
```

Eine Darstellung des Codes in **Pseudocode** wäre:

```
Zahl eingeben
Quadrat der Zahl berechnen
Quadratzahl ausgeben
```

In einem **Struktogramm** wird jede Anweisung in einem **Rechteck** dargestellt.

Eingaben werden dabei mit einem vorangestelltem **E** , Ausgaben mit einem vorangestelltem **A** gekennzeichnet. Damit ergibt sich die folgende Darstellung:

E: zahl
quadrat_zahl = zahl ** 2
A: quadrat:zahl

Das Beispiel zeigt direkt, dass die Darstellungsformen **Pseudocode** und **Struktogramm** nicht immer zu 100% zu unserer Programmiersprache passen. Im obigen Code sehen wir die folgende Zeile.

```
In [ ]: zahl = int(eingabe)
```

Diese Zeile wird in den anderen Darstellungsformen **nicht** berücksichtigt. Aber wir wissen, dass jede Nutzereingabe vom Datentyp **string** ist. Wenn wir damit rechnen wollen, **müssen** wir also den Datentyp zu einer ganzen Zahl oder Dezimalzahl ändern.

4.3. Übungen zur Sequenz

Aufgabe 1

Implementiert den folgenden Pseudocode. Testet das Programm!

```
ganze Zahl a eingeben  
ganze Zahl b eingeben  
Summe beider Zahlen berechnen  
Summe ausgeben
```

In []: *# HIER IST PLATZ FÜR EURE LÖSUNG. :)*

Aufgabe 2

Implementiert das folgende Struktogramm. Achtet bei der Ausgabe auf das exakte Setzen der Leerzeichen!

E: Zeit in Sekunden
minuten = zeit // 60
sekunden = zeit % 60
A: X Sekunden entsprechen Y minuten und Z Sekunden.

In []: *# HIER IST PLATZ FÜR EURE LÖSUNG! :)*



