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 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.

4.4. Verzweigungen

Bei Programmen und Algorithmen muss man in der Lage sein - je nach Eingabe der Nutzerinnen und Nutzer, berechneter oder eingelesener Daten - Entscheidungen zu treffen.

Je nach Entscheidung wird entweder das eine oder das andere gemacht.

Ein der Jahreszeit entsprechendes Beispiel wäre: **WENN** die Temperatur im Zimmer zu niedrig ist, **DANN** drehe die Heizung auf, **ANSONSTEN** lasse sie ausgeschaltet.

Mit **Verzweigungen** können Codeblöcke definiert werden, die nur dann ausgeführt werden, wenn eine gegebene **Bedingung erfüllt** bzw. nur dann, wenn sie **nicht erfüllt** ist.

Bedingungen sind meistens Ergebnisse von Vergleichen und damit boolesche Werte (`True` oder `False`).

Darin lässt sich bereits gut die **Struktur des Pseudocodes** einer Verzweigung erkennen. Wir nennen diese Art der Verzweigung eine **zweiseitige Verzweigung**:

```
WENN Bedingung erfüllt
DANN
    Anweisung 1
    Anweisung 2
SONST
    Anweisung 1
    Anweisung 2
```

Es ist ebenfalls möglich nur den ersten Zweig umzusetzen. Der Code wird **ausschließlich** nur dann ausgeführt, wenn eine Bedingung erfüllt ist. Dies bezeichnen wir als **einseitige Verzweigung**:

```
WENN Bedingung erfüllt
DANN
    Anweisung 1
    Anweisung 2
```

Im **Programmcode** wird dies nachfolgend an einfachen nachvollziehbaren Beispielen gezeigt.

In [ ]:
```python
zahl = 5

if zahl > 5:
    print('Die Zahl ist größer 5')
else:
    print('Die Zahl ist kleiner oder gleich 5.')
```

Es folgen noch einige Variationen für **einseitige Verzweigungen**.

Gezeigt wird dabei, dass eine Variable vom Typ `boolean` auch allein als Bedingung in der Verzweigung stehen kann, da diese ja bereits `True` oder `False` ist.

In [ ]:
```python
test = True

if test:
    print('Test 1 erfolgreich.')

if test == True:
    print('Test 2 erfolgreich.')

if not test:
    print('Test 3 erfolgreich.')
```

Für komplexere Probleme gibt es auch noch die Möglichkeit einer **mehrseitigen Verzweigung**.

Dabei kann man auf mehrere Bedingungen hin überprüfen, wie nachfolgend gezeigt.

Bei der allerersten Übereinstimmung (Abarbeitung von **oben nach unten**) werden alle nachfolgenden Bedingungen **nicht mehr** berücksichtigt.

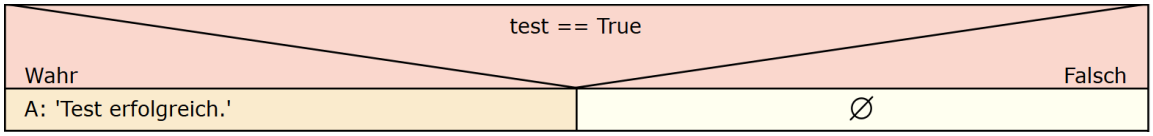
In [ ]:
```
zustand = 'super' # ändere mich!

if zustand == 'gut':
    print('Super das es dir gut geht! :)')
elif zustand == 'so lala':
    print('Schade das es dir nicht so gut geht.')
elif zustand == 'schlecht':
    print('Lass uns gern reden, wenn es dir schlecht geht!')
else:
    print('Dieses Programm kann deinen Zustand nicht verarbeiten.')
```

Hinweise zum Code einer Verzweigung:

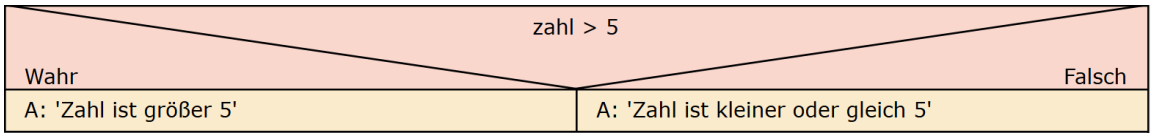
- Jede Verzweigung beginnt mit dem Schlüsselwort `if`.
- Danach folgt eine Bedingung, die wahr oder falsch sein kann. In PYTHON bedeutet dies, dass sie als Ergebnis den Wert `True` oder `False` haben muss. Nur im Falle von `True` werden die nun folgenden eingerückten Anweisungen auch abgearbeitet.
- Der Doppelpunkt `:` schließt die Bedingung ab.
- Alle Anweisungen, die bei Erfülltsein der Bedingung abgearbeitet werden sollen, müssen gleich weit eingerückt sein. Dies können auch mehrere Anweisungen sein. Nutzt Tabulatoren (die Taste mit den Doppelpfeilen am linken Rand der Tastatur) zum Einrücken.
- Die erste nicht eingerückte Anweisung kennzeichnet, dass die Verzweigung zu Ende ist. Sie wird in jedem Fall ausgeführt: Ist die Bedingung nicht erfüllt, wird sie statt der Verzweigung ausgeführt; ist die Bedingung erfüllt, wird sie erst nach Abarbeitung der Verzweigung ausgeführt.

Struktogramm für die einseitige Verzweigung:

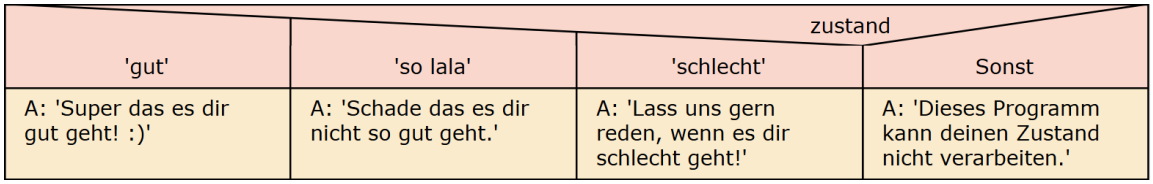


Das Zeichen `⌀` kennzeichnet eine **leere Menge**. Diese ist dem Struktogramm zwingend hinzuzufügen und bedeutet einfach, dass in diesem Fall nichts ausgeführt wird.

Struktogramm für die zweiseitige Verzweigung:



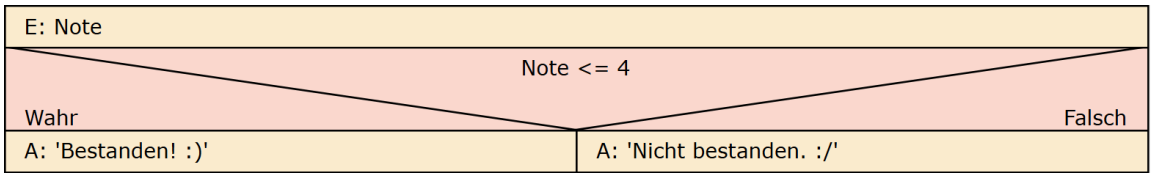
Struktogramm für die mehrseitige Verzweigung:



4.5. Übungen zu Verzweigungen

Aufgabe 1

Implementiert das folgende Struktogramm.



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

Aufgabe 2

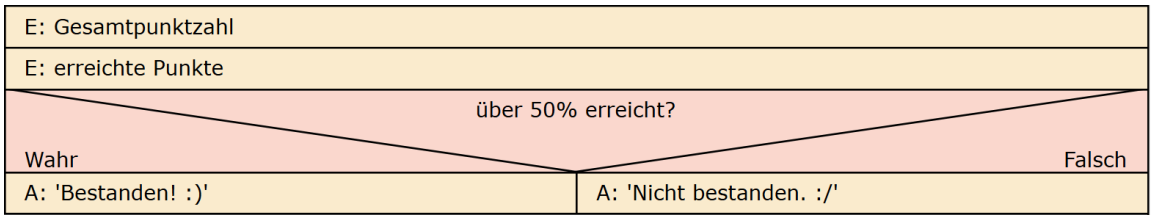
Implementiert den folgenden Pseudocode.

```
shelll
Alter als ganze Zahl eingeben
WENN Alter größer oder gleich 18
DANN
    Ausgabe von 'Zutritt gewährt.'
SONST
    Ausgabe von 'Kein Zutritt.'
Ausgabe von 'Vielen Dank für die Nutzung des Programms.'
```

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

Aufgabe 3

Implementiert das folgende Struktogramm.



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

Aufgabe 4

Implementiert ein Programm, das eine reelle Zahl einliest und ihre entgegensetzte Zahl ausgibt. D. h. bei Eingabe von z. B. −3,45 wird 3,45 ausgegeben.

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

Aufgabe 5

Implementiert ein Programm, das eine Zahl einliest und ihr Reziprokes (den Kehrwert) berechnet und ausgibt.

Hinweis: Der Kehrwert von 0 ist nicht definiert.

In [ ]:
```
# HIER IST PLATZ FÜR DEINE LÖSUNG. :)
```