L3_1_1 Die einseitige if-Anweisung / Verzweigung

Bisher haben wir Programme, die mit der ersten Anweisung starten, dann mit der zweiten Anweisung weitermachen usw. bis die letzte Anweisung ausgeführt wird. Das Programm läuft sozusagen gerade durch, ohne andere Alternativen zu berücksichtigen. Mit einer bedingten Anweisung kann man Alternativen berücksichtigen. Dies wird in Python mit einer if-Anweisung realisiert. Das untere Beispiel zeigt eine **einseitige Verzweigung**.

Beispiel:

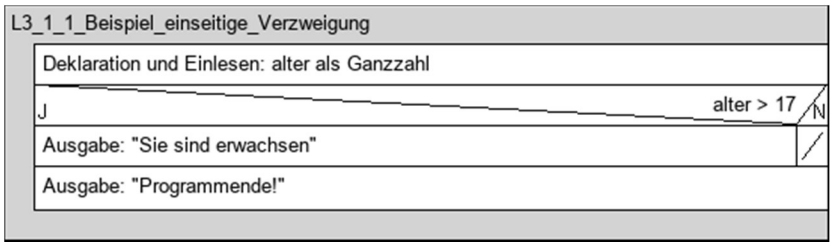
```
1  alter = int(input("Bitte geben Sie Ihr Alter in Jahren an: "))
2  if alter > 17:
3      print("Sie sind erwachsen!")
4  print("Programmende!")
```

Datei: `L3_1_1_if_einseitig.py`

Eine Variable `alter` wird deklariert und über die Benutzereingabe initialisiert. Nun wird mit der if-Anweisung eine Bedingung (alter > 17) überprüft. Diese Bedingung steht nach der if-Anweisung. Das, was das Programm machen soll, falls die Bedingung erfüllt ist, steht in dem dazugehörigen Block. Dieser Block wird in Python gebildet, indem nach der Bedingung ein Doppelpunkt kommt, der den Block einleitet (vgl. Zeile 2), und dann alle zum if-Block gehörigen Anweisungen eingerückt werden (vgl. Zeile 3). Ist der Block beendet, wird die Einrückung der Anweisungen beendet (vgl. Zeile 4).

In diesem Programm wird also in jedem Fall der Text „Programmende!“ ausgegeben, und vorher der Text „Sie sind erwachsen!“, falls die Variable `alter` einen Wert enthält, der größer ist als 17.

Struktogramm:



Ausgabe des Programms (falls alter > 17):

```
Bitte geben Sie Ihr Alter in Jahren an: 18
Sie sind erwachsen!
Programmende!
```

Wichtig:

Beachten Sie die Liste der möglichen Vergleichsoperatoren auf der folgenden Seite.

Vergleichsoperatoren

Vergleichsoperatoren werden immer zum Vergleich zweier Werte benutzt und finden ihre Anwendung zumeist beim Einsatz von Wiederholungen oder Alternativen. Als Ergebnis des Vergleichs wird ein boolescher Wert (True / False) zurückgegeben. Meist werden eine Variable mit einem festen Wert oder eine Variable mit einer anderen Variablen verglichen. Je nach Ergebnis des Vergleichs wird das Programm eine andere Reaktion zeigen.

(*Hinweis:* In Python werden auch Textvergleiche mit dem Operator „==“ durchgeführt.)

Operator	Beispiel	Beschreibung
<	a < b	a ist kleiner als b
<=	a <= b	a ist kleiner oder gleich b
>	a > b	a ist größer als b
>=	a >= b	a ist größer oder gleich b
==	a == b	a ist gleich b
!=	a != b	a ist ungleich b

L3_1_2 Die zweiseitige if-else-Anweisung / Verzweigung

Bei der einseitigen if-Anweisung wird nur der `True`-Zweig berücksichtigt. Es werden dann nur Anweisungen ausgeführt, wenn die Bedingung wahr ist. Im folgenden Beispiel gibt es auch eine Anweisung, wenn die Bedingung falsch ist.

Beispiel:

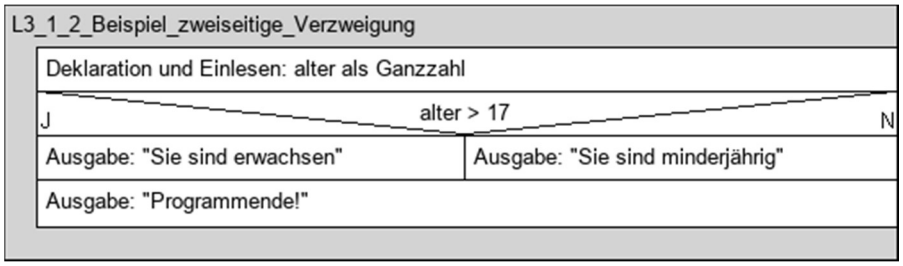
```
1 alter = int(input("Bitte geben Sie Ihr Alter in Jahren an: "))
2 if alter > 17:
3     print("Sie sind erwachsen!")
4 else:
5     print("Sie sind minderjährig!")
print("Programmende!")
```

Datei: `L3_1_2_if_zweiseitig.py`

Die if-Anweisung wird mit `else` erweitert. Nach dem `True`-Block wird der `False`-Block mit dem Schlüsselwort `else` eingeleitet. Beachten Sie auch hier den Doppelpunkt nach dem Schlüsselwort `else` sowie die darauf folgende Einrückung der Anweisungen, die zum `else`-Block gehören.

Testen Sie das Programm und variieren Sie es, bis Sie den Aufbau der Blöcke gut vor Augen haben.

Struktogramm



Ausgabe des Programms:

```
Bitte geben Sie Ihr Alter in Jahren an: 16
Sie sind minderjährig!
Programmende!
```

L3_1_3 Die geschachtelte if-Anweisung

Bei der zweiseitigen if-Anweisung gibt es zwei Alternativen bzw. Abzweigungen. Das folgende Beispiel zeigt eine Möglichkeit, wie man mehr als zwei Alternativen realisieren kann.

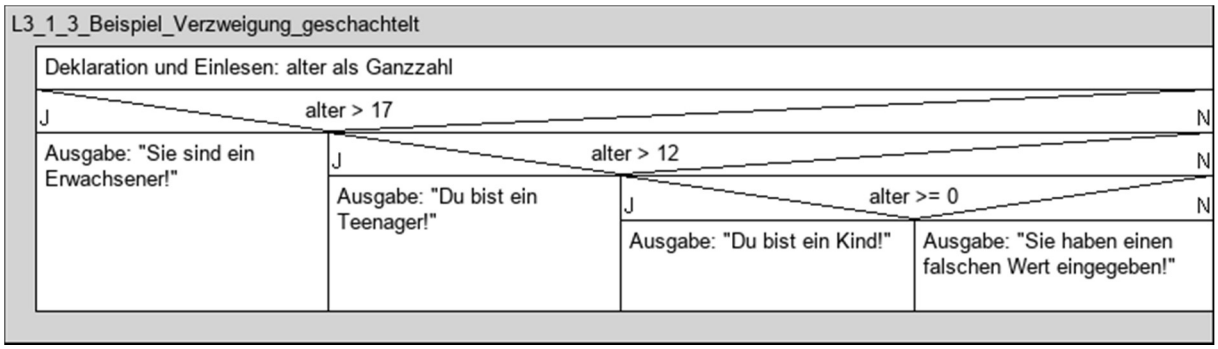
Beispiel:

```
1 alter = int(input("Bitte Alter eingeben: "))
2 if alter > 17:
3     print("Sie sind ein Erwachsener!")
4 else:
5     if alter > 12:
6         print("Du bist ein Teenager!")
7     else:
8         if alter >= 0:
9             print("Du bist ein Kind!")
10        else:
11            print("Sie haben einen falschen Wert eingegeben!")
```

Datei: L3_1_3_verzweigungen_geschachtelt.py

Im `else`-Block der ersten Bedingung wird in Zeile 5 eine zweite Bedingung mit dem Schlüsselwort `if` eingebaut (daher „*geschachtelte if-Anweisung*“). Im `else`-Block der zweiten Bedingung wird dann wiederum in Zeile 8 eine dritte Bedingung mit `if` eingegeben. Sind alle drei geprüften Bedingungen falsch, kann nur die Eingabe falsch sein (negativer Wert) und es kommt daher zu einer Fehlermeldung.

Struktogramm



Ausgabe des Programms:

```
Bitte Alter eingeben: -20
Sie haben einen falschen Wert eingegeben!
```

L3_1_4 Verknüpfte Bedingungen (logische Operatoren)

Wenn mehrere Bedingungen miteinander verknüpft werden, dann kann man die logischen Operatoren **and** für das *logische Und* bzw**. or** für das *logische Oder* nutzen. Beim *logischen Und* müssen alle Teilbedingungen wahr sein, damit die gesamte Bedingung wahr ist. Beim *logischen Oder* reicht es, wenn mindestens eine Teilbedingung wahr ist, damit die gesamte Bedingung wahr ist.

Zusätzlich gibt es den logischen Operator **not** für das *logische Nicht*. Die Wahrheitswerte einer Bedingung werden dann umgedreht. `False` wird zu `True` und `True` wird zu `False`.

Beispiel:

```
1 alter = 21
2 if alter > 12 and alter < 20:
3     print("Du bist ein Teenager!")
4
5 if alter < 10 or alter > 20:
6     print("Dein Alter liegt unter 10 ODER über 20 Jahren!")
7
8 if not alter < 18:
9     print("Dein Alter liegt NICHT unter 18 Jahren => volljährig!")
```

Datei: `L3_1_4_Verknüpfte_Bedingungen.py`

Zeile 1: Die Variable `alter` wird deklariert und erhält den Wert 21.

Zeile 2: Hier wird geprüft, ob `alter` größer 12 **UND** kleiner 20 ist. Falls das der Fall ist, kommt die dazu passende Ausgabe.

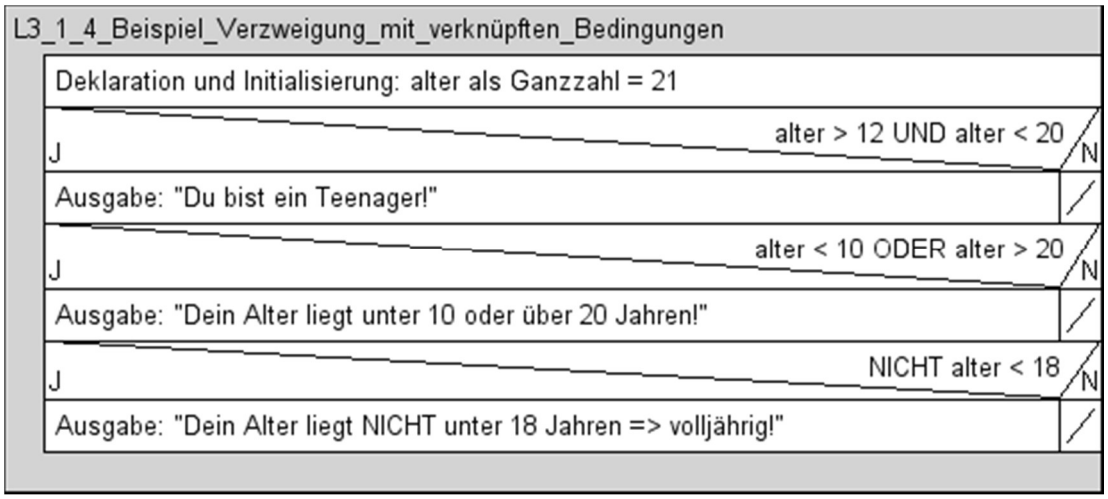
Zeile 5: Hier wird geprüft, ob `alter` kleiner zehn **ODER** größer 20 ist. Falls das der Fall ist, kommt die dazu passende Ausgabe.

Zeile 8: Hier wird festgestellt, ob `alter` **NICHT** kleiner 18 ist. Falls diese Aussage erfüllt ist, kommt die dazu passende Ausgabe.

Entsprechend erzeugt das Programm folgende Ausgabe:

```
Dein Alter liegt unter 10 ODER über 20 Jahren!
Dein Alter liegt NICHT unter 18 Jahren => volljährig!
```

Struktogramm:



Denkaufgabe / Test yourself:

Was wird das Programm ausgeben, wenn die Variable `alter` zu Beginn des Programms auf 17 gesetzt wird? Öffnen Sie das obige Programm und überprüfen Sie, ob Ihre Vorhersage stimmt.

Übersicht über die logischen Operatoren

Hier eine kurze Übersicht über die Verwendung von Operatoren in Python:

Operator	Beispiel	Beschreibung	Erläuterung
and	alter > 12 **and** alter < 20	und	Die Bedingung ist erfüllt (True), wenn beide Aussagen zutreffen.
or	alter < 10 **or** alter > 20	oder	Die Bedingung ist erfüllt (True), wenn die eine **oder** die andere Aussage (oder beide) erfüllt sind.
not	**not** alter < 18	nicht	**not** ändert einen Ausdruck von True in False und umgekehrt. Hier ist die Bedingung erfüllt, wenn das Alter nicht < 18 ist.

L3_2_1 Die for-Schleife (Zählerschleife)

Teil 1: Vorbereitung

1.1 Einfache Listen erstellen

In Python kann mit der Anweisung `range(endwert)` eine Liste mit Zahlenwerten angelegt werden. Dabei wird eine Liste erzeugt, die bei 0 beginnt und bis zur letzten Zahl vor *endwert* reicht, also von 0 bis *endwert-1*. Die im folgenden Beispiel mit `range(10)` erzeugte Zahlenliste beginnt also mit 0 und geht bis 9, nämlich *endwert - 1*, d.h. 10 - 1 = 9:

```
1 zahlenliste = range(10)
2 # Ausgabe der Liste mit ihren Zahlenwerten:
3 print(list(zahlenliste))
```

Datei: L3_2_1_a_range-Anweisung.py

Ergebnis des Programms:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Hier sieht man, dass die erzeugte Liste mit 0 beginnt und dann bis zur letzten Zahl vor 10, also bis 9, reicht (denn 9 = 10 – 1).

Analog erzeugt die Anweisung range(5) eine Liste von 0 bis 4:

```
1 zahlenliste = range(5)
2 # Ausgabe der Liste mit ihren Zahlenwerten:
3 print(list(zahlenliste))
```

Datei: L3_2_1_b_range-Anweisung.py

Ergebnis des Programms:

```
[0, 1, 2, 3, 4]
```

1.2 Listen mit einem eigenen Startwert erstellen

Man kann der Liste aber auch einen anderen Startwert als 0 geben, dazu wird die Anweisung `range(anfangswert, endwert)` benutzt:

```
1 zahlenliste = range(3, 10)
2 # Ausgabe der Liste mit ihren Zahlenwerten:
3 print(list(zahlenliste))
```

Datei: L3_2_1_c_range-Anweisung.py

Ergebnis des Programms:

```
[3, 4, 5, 6, 7, 8, 9]
```

1.3 Listen mit einer anderen Schrittweite erstellen

Möchte man zusätzlich eine andere Schrittweite verwenden, wird hierfür die Anweisung `range(anfangswert, endwert, schrittweite)` benutzt:

```
1 zahlenliste = range(1, 16, 2)
2 # Ausgabe der Liste mit ihren Zahlenwerten:
3 print(list(zahlenliste))
```

Datei: `L3_2_1_d_range-Anweisung.py`

Ergebnis des Programms:

```
[1, 3, 5, 7, 9, 11, 13, 15]
```

1.4 Aufgaben

a) Ändern Sie das Programm so ab, dass das Programm eine Liste von 0 bis 15 erzeugt.

b) Ändern Sie das Programm so ab, dass das Programm eine Liste von 8 bis 19 erzeugt.

c) Ändern Sie das Programm so ab, dass das Programm eine Liste aller geraden Zahlen von 2 bis 20 erzeugt.

d) Ändern Sie das Programm so ab, dass das Programm eine Liste in absteigender Reihenfolge von 10 bis 1 erzeugt.

Hinweis: Die Lösungen zu diesen Aufgaben finden Sie in der Datei `L3_2_1_e_range-Anweisung_Zusatzbeispiele.py`

Teil 2: Die for-Schleife

In manchen Fällen muss ein Programm immer wieder die gleichen Anweisungen hintereinander ausführen, bis ein bestimmtes Ergebnis eintritt. Wenn der Programmierer schon vorher weiß, wie oft ein Programm wiederholt werden muss, wird oft die **for-Schleife** benutzt.

Das folgende Programm gibt zeilenweise alle Zahlen von 0 bis 9 aus:

Beispiel:

```
1 for zahl in range(10):
2     print(zahl)
```

Datei: L3_2_1_for_schleife.py

In Python besteht die for-Schleife aus drei Elementen:

1. Dem Schlüsselwort „for“, mit dem die Schleife eingeleitet wird,
2. einer Variablen (hier: *zahl*), die mit in der Reihe nach alle Werte der folgenden Liste annimmt,
3. einer Liste, in der die Werte stehen, die die angegebene Variable der Reihe nach annimmt. Diese Liste kann mit range(*endwert*) angegeben werden. Nach der Angabe der Liste folgt ein Doppelpunkt als Zeichen dafür, dass hier der Inhalt der Schleife beginnt. In den folgenden Zeilen stehen die Anweisungen, die im Rahmen der Schleife ausgeführt werden sollen, wobei die Zeilen, die zur Schleife gehören, eingerückt sein müssen. Dabei nimmt *zahl* der Reihe nach alle Werte in der Liste an, d.h. im obigen Beispiel nimmt *zahl* im ersten Durchlauf den Wert 0 an, im zweiten Durchlauf den Wert 1, danach 2 und so weiter, bis *zahl* im letzten Durchlauf den letzten Wert der Liste annimmt, also 9.

Anwendungsbeispiel:

Es soll ein Programm geschrieben werden, das die 5er-Reihe von 5 bis 50 ausgibt:

```
1 produkt=0
2 for zahl in range(1, 11):
3     produkt = zahl * 5
4     print(zahl, "mal 5 =", produkt)
```

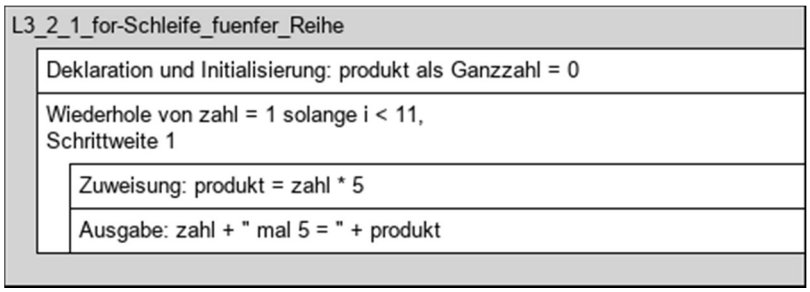
Datei: L3_2_1_for_Schleife_fuenfer_Reihe.py

In diesem Programm nimmt die Variable *zahl* der Reihe nach die Werte von 1 bis 10 an. Bei jedem Schleifendurchlauf wird *zahl* mit 5 multipliziert und anschließend auf dem Bildschirm ausgegeben, beim ersten Durchgang also 1 mal 5, beim zweiten Durchgang 2 mal 5 etc., bis zum Schluss dann 10 mal 5 berechnet und ausgegeben wird.

Ausgabe des Programms:

```
1 mal 5 =  5
2 mal 5 =  10
3 mal 5 =  15
4 mal 5 =  20
5 mal 5 =  25
6 mal 5 =  30
7 mal 5 =  35
8 mal 5 =  40
9 mal 5 =  45
10 mal 5 =  50
```

Struktogramm:



L3_2_2 While-Schleife

In manchen Fällen muss ein Programm immer wieder die gleichen Anweisungen viele Male hintereinander ausführen, bis ein bestimmtes Ergebnis eintritt. Wie oft man diese Anweisungen wiederholen muss, hängt dabei von einer Bedingung ab, welche am Anfang, bevor die Anweisungen ausgeführt werden, überprüft wird. Solche Wiederholungen nennt man auch Schleifen.

Ist die Anzahl der Schleifendurchläufe vor dem Beginn des Programms nicht exakt bekannt, benutzt man eine **while-Schleife**.

Beispiel 1:

Das folgende Programm simuliert einen Würfelvorgang und würfelt solange, bis eine 6 gewürfelt wurde:

```
1 from random import *
2
3 zufallszahl = 0
4 while zufallszahl != 6:
5     zufallszahl = randint(1, 6)
6     print ("Wurf:", zufallszahl)
```

Datei: `L3_2_2_a_while_Schleife.py`

Hinweis: *In diesem Programm kommen zwei Anweisungen vor, die benötigt werden, weil mit Zufallszahlen gearbeitet werden soll (nicht abiturrelevant!):*
In Zeile 1 wird ein Modul in das Programm eingebunden, das Zufallszahlenfunktionalität zur Verfügung stellt, in Zeile 4 wird dann eine Funktion des Moduls genutzt, die Funktion `randint(1, 6)`, die eine ganzzahlige Zufallszahl aus dem Bereich 1 bis 6 erzeugt.

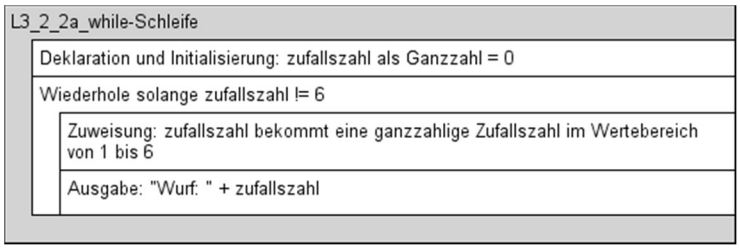
Erklärung des Programmablaufs:

In Zeile 3 erhält die Variable `zufallszahl` den Wert 0. Dann wird in einer while-Schleife der Variablen `zufallszahl` solange ein Zufallswert im Wertebereich von 1 bis 6 zugewiesen, bis `zufallszahl=6` ist. Bei jedem Durchlauf wird der jeweilige Wert von `zufallszahl` ausgegeben.

Aufbau der while-Schleife in Python:

Eine while-Schleife wird in Python mit dem Schlüsselwort `while` eingeleitet. Danach folgt die Bedingung, die erfüllt sein muss, damit die Schleife wiederholt wird, d.h. die Schleife wird wiederholt, solange die Bedingung erfüllt ist (hier im Beispiel: solange `zufallszahl` ungleich 6 ist). Abschließend folgt ein Doppelpunkt, und unter dem Doppelpunkt stehen **eingerückt** alle Anweisungen, die im Rahmen der Schleife ausgeführt werden sollen.

Struktogramm:



Mögliche Ausgabe (je nach Würfelergebnis):

```
Wurf: 5
Wurf: 3
Wurf: 5
Wurf: 6
```

Beispiel 2:

Andererseits lässt sich jede for-Schleife auch in eine while-Schleife übertragen. Wir vergleichen das Beispiel der for-Schleife aus der letzten Einheit mit der Umsetzung als while-Schleife.

Anwendungsbeispiel: Es soll ein Programm geschrieben werden, das die 5er-Reihe von 5 bis 50 ausgibt:

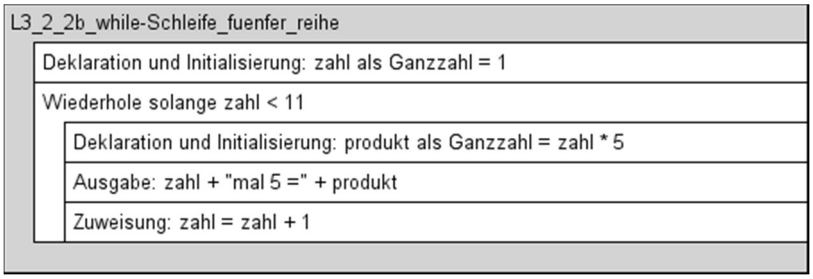
Quellcode:

```
1 for zahl in range(1, 11):
2     produkt = zahl * 5
3     print(zahl, "mal 5 =", produkt)
```

```
1 zahl = 1
2 while zahl < 11:
3     produkt = zahl * 5
4     print(zahl, "mal 5 =", produkt)
5     zahl = zahl + 1
```

Datei: L3_2_2_b_while_Schleife.py

Im Vergleich ist ein wesentlicher Unterschied festzustellen:

In der for-Schleife greifen wir auf eine festgelegte Zahlenliste zurück. Diese nutzen wir bei der while-Schleife nicht. Daher muss dort der Zähler zunächst deklariert und initialisiert werden. Ebenso muss in der while-Schleife der Zähler manuell verändert werden.

Struktogramm: