Repetition Programmieren Python

Dr. Philipp Hurni, Kantonsschule Sursee

Agenda

Selbständiges Arbeiten:
Repetition und Übungen zu

- Variablen und Berechnungen
- Schleifen
- Input/Output
- Zufallszahlen
- Listen
- Funktionen

Grundanforderungen Programmieren

- **Umgang mit Variablen**
  - Rechnen
  - Zuweisen
  - Ausgeben
- **For-Schlaufe / While-Schlaufe**
  - For Schlaufe verstehen und verwenden
  - Laufvariablen verstehen
  - Bedingungen in While Schlaufe verstehen und einsetzen können
- **Input / Print**
  - Einen Input einlesen
  - Input zu Zahl konvertieren
- **Zufallszahlen**
  - Generieren mit randint(x,y)
- **Umgang mit Listen**
  - Direkte Zuweisung eines Elements
  - Hinzufügen von Elementen
- **Funktionen**
  - Lesen und Interpretieren
  - Abändern, Erstellen
  - Prinzip Rekursion verstehen
- **If-Else Verzweigung**
  - Lesen und interpretieren
  - Abändern, Erstellen

Variablen und Berechnungen

Variablen und Berechnungen

Was sind Variablen?

- Grundsätzlich ist eine Variable ist ein Behälter, der einen Wert speichert. Man kann auch von "Bezeichner" sprechen.
- Die Syntax dabei: ein einfaches Gleichzeichen bezeichnet die Zuweisung eines Werts zu einerm Variablenname (variablenname = wert)
  Beispiele:
  - x = "hallo" (die Variable mit dem Namen x hat nun den Wert "hallo"
  - y = 12.5 (die Variable mit dem Namen y hat nun den Wert y

Variablen und Berechnungen

- Regeln für Variablennamen:
  - Dürfen nur Buchstaben (a-z, A-Z), Zahlen (0-9) und Unterstriche (_) enthalten
  - Dürfen nicht mit einer Zahl beginnen
  - Sind case-sensitive (mein_alter ist nicht gleich Mein_Alter)
- Beispiele
  - banane123 = 1450 (gültig)
  - 123banane = 1245 (ungültig)
  - albert_roesti = 49 (gültig)
  - _albert_roesti = 49 (gültig)
  - albert_roesti! = 49 (ungültig)
  - _ = 49 (gültig)

Variablen und Berechnungen

Datentypen:

String (Text):	name = "Max"
Integer (Ganzzahl):	alter = 25
Float (Gleitkommazahl):	preis = 19.99
Boolean (Wahrheitswert):	ist_gluecklich = True

Gleitkommazahlen

Float: Dies ist der gebräuchlichste Begriff in vielen Programmiersprachen (z.B. Python, C++, Java), um einen Datentyp für Gleitkommazahlen zu beschreiben. Abkürzung für "Floating Point Number".

Kommazahl: Eine umgangssprachliche Bezeichnung, die das Vorhandensein eines Dezimalkommas (oder -punkts) hervorhebt.

Andere Bezeichnungen für genau dasselbe:
Gleitpunktzahl, Fliesskommazahl, Fliesskommazahl

Rechenoperatoren in Python

Operator	Name	Funktion
a + b	Addition	Addiert zwei Werte a und b
a - b	Subtraktion	Subtrahiert den rechten Wert b vom linken a
a * b	Multiplikation	Multipliziert zwei Werte a und b
a ** b	Potenz	Berechnet die Potenz, der linke Wert ist die Basis, der rechte der Exponent a hoch b
a / b	Division	Dividiert den linken Wert durch den rechten und gibt immer einen Float zurück (!)
a // b	Ganzzahl Division	Teilt a durch b ganzzahlig – Resultat ist immer ein Integer
a % b	Modulo (Rest-Operator)	Beispiel: 10 % 3 wäre 1
(a+b)*c	() bestimmt die Reihenfolge der Operationen	Genau wie in der Mathematik (1+2)*3 ergibt 9

Aufgabe A

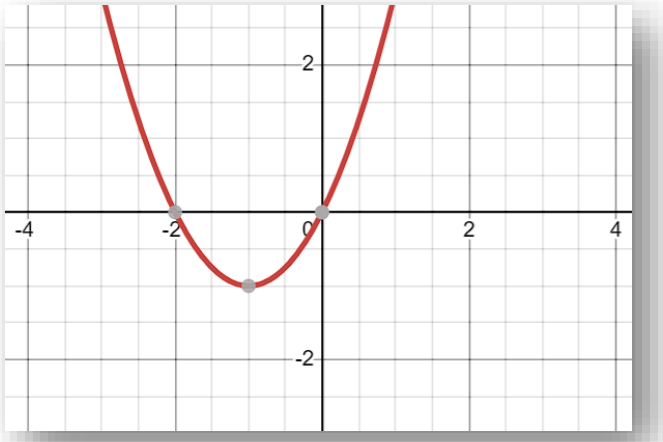
Berechne die Variablen x1 und x2 anhand dieser (Zauber)Formel

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Setze a=1, b=2, c=0

Lösung A

```
a=1
b=2
c=0
x1 = (-b + (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
x2 = (-b - (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
print(x1, x2)
```



0, -2

Aufgabe B

a. age
b. ?age
c. Age
d. 1age
e. _age
f. age_
g. age_*
h. +age

Welches sind ungültige Variablennamen – und warum?

Lösung B

a. age
b. ?age
c. Age
d. 1age
e. _age
f. age_
g. age_*
h. +age

Aufgabe C: Modulo und Rest

Wenn zur Zeit 13:37 Uhr ist, wie viel Uhr ist es in 1000000 Minuten

Lösung C: Modulo und Rest

Wenn zur Zeit 13:37 Uhr ist, wie viel Uhr ist es in 1000000 Minuten

```
h=13
m=37

m = (m + 1000000) % 60
h = (h + (1000000 // 60)) % 24

print(m)
print(h)
```

Schlaufen, Schleifen

Was ist eine for-Schleife?

Was ist eine for-Schleife?
Wiederholung von Anweisungen für eine bestimmte Anzahl von Malen.

Einfache range()-Funktion: range(stop) – Zählt von 0 bis **stop-1**.

Achtung: Einrückung definiert, bis wo was zur Schleife gehört!

```
for i in range(5):
        print(i)
```

Einrückung

```
# Ausgabe: 0, 1, 2, 3, 4
```

Was ist eine for-Schleife?

For-Schleife kann ich auch auf Listen anwenden:

```
fruechte = ["Apfel", "Banane", "Kirsche"]
for frucht in fruechte:
    print(frucht)
```

Aufgabe D

Schreibe eine Schlaufe, welche folgendes Zahlenmuster ausgibt:

0
1
3
6
10
15
21
28
36
45

Lösung D

```
summe=0
for i in range(10):
    summe = summe + i
    print(summe)
```

Aufgabe E

Schreibe eine Schlaufe, welche folgendes Zahlenmuster ausgibt:

1
2
3
5
8
13
21
34

Quizfrage: wie heisst diese Zahlenfolge? Wie kann man sie beschreiben?

Lösung E

```
# Initialisiere die ersten beiden Zahlen der Folge
a = 1
b = 2

print(a)
print(b)

# berechne die nächsten 10 Glieder
for i in range(10):
    # Speichere den Wert von b in einer temporären Variable
    temp = b
    # Berechne den neuen Wert von b
    b = a + b
    # Aktualisiere a mit dem alten Wert von b
    a = temp
    # Gib den neuen Wert von b aus
    print(b)
```

Name der Zahlenfolge:
Fibonacci-Reihe

Print und Input

Print() Funktion

Die print()-Funktion gibt Werte auf der Konsole aus.

Man kann ihr einen oder mehrere Werte (Variablen, Strings, Zahlen etc.) übergeben. Mehrere Werte werden standardmässig durch ein Leerzeichen voneinander getrennt ausgegeben.

```
alter = 15
print("Ich bin", alter, "Jahre alt.")
```

Input Funktion

Mit der input()-Funktion kann man Benutzereingaben über die Konsole entgegennehmen.

Achtung: Die Funktion gibt **immer einen String (Text)** zurück, auch wenn der Benutzer eine Zahl eingibt.

Der Text, der als Argument übergeben wird, dient als Aufforderung für den Benutzer (Prompt). Beispiel:

```
namen = input("Gib deinen Namen ein: ")
print("Hallo ", namen, " du hast einen schönen Namen!" )
```

Typumwandlung mit int() und float()

int(): Wandelt einen Wert in eine ganze Zahl (Integer) um. Wenn die Eingabe eine Gleitkommazahl ist, werden die Nachkommastellen abgeschnitten (es wird nicht gerundet!). Wandelt auch Strings um, solange sie nur Ziffern enthalten.

int(3.14) wird zu 3.
int(-5.9) wird zu -5.
int("123") wird zu 123.

int("Hallo") führt zu einem Fehler (ValueError).

Typumwandlung mit int() und float()

float(): Wandelt einen Wert in eine Gleitkommazahl (Float) um.
Wandelt Ganzzahlen um, indem es eine .0 anfügt. Wandelt auch Strings um, solange sie eine gültige Zahlendarstellung haben.

float(5) wird zu 5.0.
float("3.14") wird zu 3.14.
float("100") wird zu 100.0.
float("Python") führt zu einem Fehler (ValueError).

Wichtig: Achte immer darauf, dass der Wert, den du umwandeln möchtest, auch tatsächlich in den Ziel-Datentyp konvertiert werden kann. Andernfalls bricht das Programm mit einem ValueError ab.

Aufgabe F

Schreibe ein Programm, welches Franken zu Euro konvertiert

```
wechselkurs = 0.95
franken_input = input("Wechselkursrechner – gib Franken ein: ")

???
```

Lösung F

Schreibe ein Programm, welches Franken zu Euro konvertiert

```
wechselkurs = 0.95
franken_input = input("Wechselkursrechner – gib Franken ein : ")
franken_input = float(franken_input) # umwandlung in float

euro_ergebnis = franken_input * wechselkurs
print("Franken:", franken_input, "entsprechen", euro_ergebnis, " Euro")
```

Aufgabe G

Schreibe ein Programm, welches eine Zufallszahl generiert, und fragt ob es weiterfahren soll, bis man "Nein" eingibt.

Lösung G

```
from random import *

zufallszahl = randint(0,100)
weiterfahren_antwort = "Ja"

while(weiterfahren_antwort != "Nein"):
    zufallszahl = randint(0,100)
    print("Zufallszahl: ", zufallszahl)
    weiterfahren_antwort = input("soll ich weiterfahren")

print("schönen Tag noch!")
```

Umgang mit Listen

Listen (Lists)

Listen (Lists)

Definition: Eine Liste ist eine geordnete Sammlung von Werten, die verschiedene Datentypen enthalten können. Listen sind veränderbar (mutable), d.h. man kann ihre Elemente hinzufügen, entfernen oder ändern.

Erstellung: Listen werden mit eckigen Klammern [] erstellt, wobei die Elemente durch Kommas getrennt sind.

Python

```
meine_liste = ["Apfel", 42, 3.14, True]
```

Listenzugriff und Änderung

Zugriff: Du greifst auf einzelne Elemente über ihren Index zu. Der Index beginnt bei 0. Du kannst die Werte an bestimmten Indizes ändern.

```
meine_liste = ["Apfel", 42, 3.14, True]
print(meine_liste[0])
# Ausgabe: Apfel

meine_liste[1] = "Banane"
print(meine_liste)
# Ausgabe: ['Apfel', 'Banane', 3.14, True]
```

Listenmethoden append, remove, pop

Listen haben nützliche Methoden, um Elemente zu manipulieren:
.append(element): Fügt ein Element am Ende hinzu.
.remove(element): Entfernt das erste Vorkommen eines bestimmten Elements.
.pop(index): Entfernt und gibt das Element am angegebenen Index zurück.

```
meine_liste.append("Orange")
meine_liste.remove("Apfel")
print(meine_liste)
```

Ausgabe: ['Banane', 3.14, True, 'Orange']

Aufgabe H

Schreibe ein Programm, welches in der folgenden Liste die grösste und die kleinste Zahl sucht

zahlen = [0,14,5,6,7,8,9,11,21,34,56,78,2,1,3]

Tip: Die Länge der Liste zahlen berechnet man mit der Funktion **len(**zahlen**)**

Lösung H

```
zahlen = [0,14,5,6,7,8,9,11,21,34,56,78,2,1,3]

groesste = zahlen[0]
kleinste = zahlen[0]

for i in range(0, len(zahlen)):
    if zahlen[i] > groesste:
        groesste = zahlen[i]

    if zahlen[i] < kleinste:
        kleinste = zahlen[i]

print("groesste: ", groesste)
print("kleinste: ", kleinste)
```

Aufgabe I

tellen Sie sich vor, Sie verwalten eine digitale Einkaufsliste.

Erstellen Sie eine Liste mit dem Namen **einkaufsliste**, die die folgenden drei Elemente enthält: "Milch", "Brot", "Äpfel".

Fügen Sie das Element "Käse" am Ende der Liste hinzu. Verwenden Sie dafür die Methode append().

Sie haben bereits "Brot" gekauft. Entfernen Sie es aus der Liste mit der Methode remove().

Geben Sie am Ende die aktualisierte **einkaufsliste** aus.

Lösung I

```
# 1. Erstellen Sie die Einkaufsliste
einkaufsliste = ["Milch", "Brot", "Äpfel"]
# 2. Fügen Sie "Käse" hinzu
einkaufsliste.append("Käse")
# 3. Entfernen Sie "Brot"
einkaufsliste.remove("Brot")
# 4. Geben Sie die Liste aus
print(einkaufsliste)
```

Funktionen

Funktionen

Eine Funktion ist ein Code-Block, der nur ausgeführt wird, wenn er aufgerufen wird. Sie wird verwendet, um Code zu organisieren und wiederverwendbar zu machen. Du **definierst** eine Funktion mit dem Schlüsselwort **def**, gefolgt von einem Namen, runden Klammern (), je nachdem Parametern und einem Doppelpunkt – z.B. addiere(a,b)

```
def addiere(a, b):
        ergebnis = a + b
        return ergebnis

print(addiere(2,2))
```

Definition der Funktion (def)

Aufruf der Funktion

Um den Code innerhalb einer Funktion auszuführen, musst du die Funktion **aufrufen**, indem du ihren Namen gefolgt von den runden Klammern () schreibst.

Funktionen mit Parametern und Rückgabewert

```
# Funktionsdefinition mit Parametern 'a' und 'b'
def addiere(a, b):
        ergebnis = a + b
        return ergebnis # Rückgabe des Ergebnisses

# Funktionsaufruf
summe = addiere(5, 3)

# Die Werte 5 und 3 sind die Argumente
print(summe) # Ausgabe: 8
```

Funktionen mit Parametern und Rückgabewert

Parameter: Parameter sind Platzhalter in der Funktionsdefinition. Sie erlauben es, der Funktion Daten zu übergeben, damit sie damit arbeiten kann. Du kannst dir Parameter wie die Zutaten für ein Rezept vorstellen. Die Funktion ist das Rezept, und die Parameter sind die Zutaten, die du brauchst, um das Gericht zuzubereiten.

Beispiel: In def addiere(a, b): sind a und b die Parameter.

Rückgabewert (return): Ein Rückgabewert ist das Ergebnis, das eine Funktion nach ihrer Ausführung zurückgibt.
Du kannst dir den Rückgabewert wie das fertige Gericht aus dem Rezept vorstellen. Die Funktion hat ihre Arbeit getan und gibt das Ergebnis zurück, damit du es weiterverwenden kannst. Das Schlüsselwort return wird verwendet, um einen Wert zurückzugeben.

Beispiel: return a + b gibt das Ergebnis der Addition an den Aufrufer zurück.

Aufgabe J

Schreibe ein Programm, welches die Argumente der Funktion mult miteinander multipliziert und zurückgibt

```
def mult(x,y,z):
        …

mult(2,3,4)
```

Lösung J

Schreibe ein Programm, welches die Argumente der Funktion mult miteinander multipliziert und ausgibt

```
def mult(x,y,z):
    return (x*y*z)

rint(mult(2,3,4))
```

Aufgabe K

Schreibe eine Funktion mult, welches eine Liste übernimmt und die Zahlen miteinander multipliziert und ausgibt

```
liste = [2,3,4]
def mult(liste):
        …
```

Aufgabe K

Schreibe eine Funktion mult, welches eine Liste übernimmt und die Zahlen miteinander multipliziert und ausgibt

```
liste = [2,3,4]
def mult(liste):
    m=1
    for i in liste:
        m = m * i
    return m

print(mult(liste))
```

Aufgabe L

Schreibe eine Funktion, welche die Fibonacci-Reihe produziert. Diesmal unter Verwendung von Funktionen

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

$$f(n) = \begin{cases} n = 0 & 0 \\ n = 1 & 1 \\ n > 1 & f(n-1) + f(n-2) \end{cases}$$

Lösung L – Mit Liste

```
def fibonacci_reihe_anzahl(anzahl_glieder):
  # Initialisiert die ersten beiden Zahlen der Folge
  a = 0
  b = 1

  # Erstellt eine leere Liste, um die Zahlen zu speichern
  ergebnis_liste = []

  # Fügt die ersten beiden Zahlen hinzu, falls die Anzahl groß genug ist
  if anzahl_glieder >= 1:
      ergebnis_liste.append(a)
  if anzahl_glieder >= 2:
      ergebnis_liste.append(b)

     # Berechnet die restlichen Glieder mit einer for-Schleife
  for i in range(2, anzahl_glieder):
      temp = a + b  # Berechnet die nächste Zahl
      ergebnis_liste.append(temp)
      a = b         # Aktualisiert a mit dem alten Wert von b
      b = temp      # Aktualisiert b mit der neu berechneten Zahl

  return ergebnis_liste

# Beispiel für den Aufruf der Funktion
anzahl = 10
ergebnis = fibonacci_reihe_anzahl(anzahl)
print("Fibonacci-Reihe mit ", anzahl, "Gliedern: ", ergebnis)
```

Lösung L - Rekursiv

```
def fibonacci_rekursiv(n):
   if n <= 1:
      return n
   else:
      return(fibonacci_rekursiv(n-1) + fibonacci_rekursiv(n-2))

anzahlTerme = 10

print("Fibonacci Reihe:")
for i in range(anzahlTerme):
    print(fibonacci_rekursiv(i))
```

If Else Verzweigungen

Bedingte Anweisungen: if, elif, else

Bedingte Anweisungen werden verwendet, um Code nur dann auszuführen, wenn bestimmte Bedingungen erfüllt sind. Sie ermöglichen es deinem Programm, Entscheidungen zu treffen.

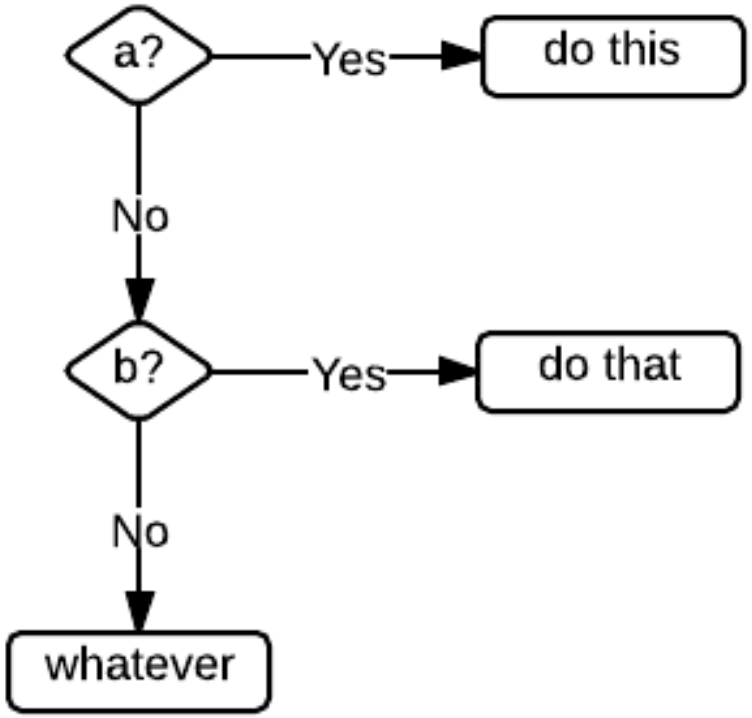
if: Der Code in diesem Block wird ausgeführt, wenn die Bedingung als True bewertet wird. Es ist der Start jeder bedingten Anweisung.

elif (Else If): Dies steht für "andernfalls, falls". Der Code in diesem Block wird ausgeführt, wenn die vorherige if-Bedingung (oder elif-Bedingung) False war und diese neue elif-Bedingung True ist. Du kannst beliebig viele elif-Blöcke haben.

else: Der Code in diesem Block wird ausgeführt, wenn alle vorhergehenden if- und elif-Bedingungen False waren. Ein else-Block ist optional.

If – elif – else

```
if a:
    do this
elif b:
    do that
else:
    whatever
```



Bedingte Anweisungen: if, elif, else

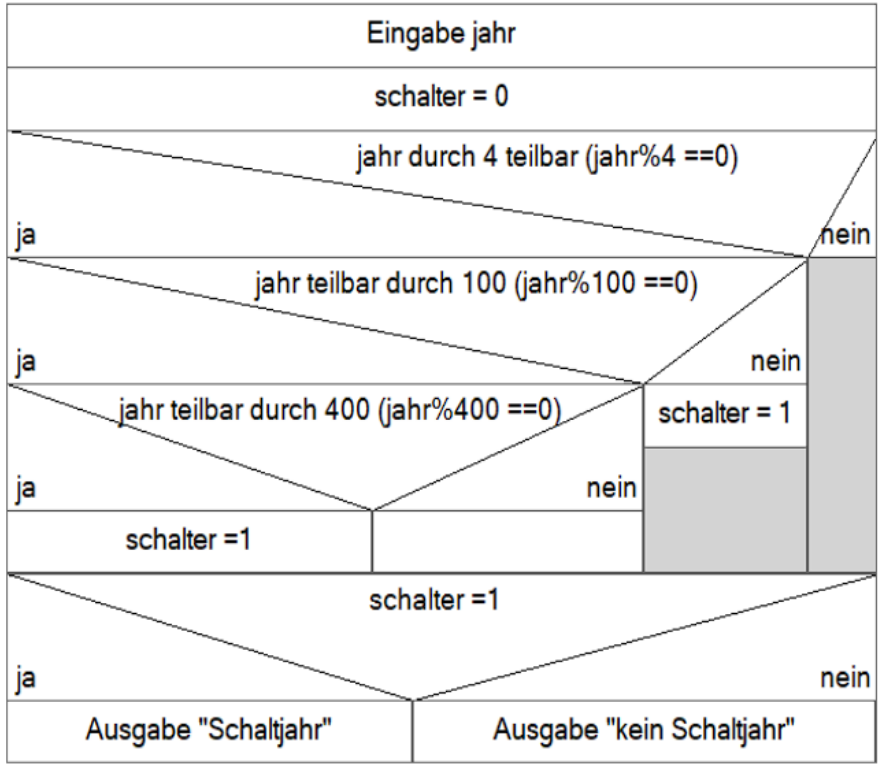
```
note = 4

if note == 6:
        print("Sehr gut")
elif note == 5:
        print("Gut")
elif note == 4:
        print("Genügend")
else:
        print("Nicht bestanden")

# Ausgabe: Genügend
```

Aufgabe M

Schreibe ein Programm, welches für ein gegebenes Jahr angibt, ob es sich um ein Schaltjahr handelt oder nicht



Lösung M

```
jahr = 2023
schalter = 0

if (jahr % 4 == 0 and jahr % 100 != 0) or (jahr % 400 == 0):
        schalter = 1
else:
        schalter = 0

if(schalter==1):
    print("Schaltjahr")
else:
    print("kein Schaltjahr")
```

Aufgabe N

Definiere eine Funktion in Python welche 3 Werte übernimmt und das Maximum berechnet und wieder ausgibt

Beispiel
Input:
a = 30
b = 22
c = 18

Erwartetes Resultat
30 ist das Maximum der Inputwerte

Lösung N

```
def max_of_three(a, b, c):
        if a >= b and a >= c:
                print(a, "ist das Maximum der Inputwerte")
        elif b >= a and b >= c:
                print(b, "ist das Maximum der Inputwerte")
        else:
                print(c, "ist das Maximum der Inputwerte")

max_of_three(30, 22, 18)
```

While Schlaufen

While Schleife

Eine while-Schleife führt einen Code-Block so lange aus, wie eine Bedingung True ist. Sie ist ideal, wenn du nicht im Voraus weisst, wie oft die Schleife durchlaufen werden soll.

Syntax: Die Schleife beginnt mit dem Schlüsselwort while, gefolgt von einer Bedingung und einem Doppelpunkt.

Bedingung: Der Code im Schleifenkörper wird wiederholt, solange die Bedingung True bleibt. Sobald sie False wird, springt das Programm aus der Schleife.

Vorsicht! Endlosschleife: Du musst sicherstellen, dass die Bedingung irgendwann False wird, sonst läuft die Schleife unendlich. Das passiert oft, wenn man die Variable, die in der Bedingung verwendet wird, im Schleifenkörper nicht aktualisiert.

While Schleife

```
zaehler = 0
while zaehler < 4:
        print(zaehler)
        zaehler = zaehler + 1

# Wichtig: Der Zähler wird erhöht, damit die Schleife endet
# Ausgabe:
# 0
# 1
# 2
# 3
# 34
```

Aufgabe O

Berechne die Fakultätsfunktion – mit einer while Schlaufe und unter Verwendung von Funktionen

Man soll eingeben können, für welche Zahl man die Fakultät berechnen will (z.B. 6!) – und dann wird einem die Fakultät berechnet und ausgegeben (z.B. 720 für 6!)

Lösung O

```
def factorial(n):
    n_input = n
    fact = 1
    while(n!=0):
        fact *= n
        n = n-1
    return fact

inputNumber = int(input("Für welche Zahl?: "))
print(factorial(inputNumber))
```