LÖSUNGSVORSCHLÄGE ZU DEN FINGERÜBUNGEN

Bei Fragen wenden Sie sich bitte an Ihre Tutoren
(z. B. über das **Piazza**-Forum oder *s.berktold@tum.de*)

Kontrollflussdiagramm

```
print("Gib eine natürliche Zahl ein: ")
i = int(input())

if i >= 0: # gültige Eingabe
    result = 1 # definiere das Ergebnis
    while i > 0: # führe Berechnung durch
        result = result*i
        i -= 1
    print("Ergebnis: " + str(result))
else: # negative Eingabe
    print("Ungültige Eingabe!")
```

Das Programm berechnet die **Fakultät** der vom Benutzer eingegebenen Zahl `i`. Da die Fakultät von negativen Zahlen nicht definiert ist, bricht das Programm mit einer entsprechenden Fehlermeldung ab, falls eine negative Zahl eingegeben wird. Das Ergebnis wird standardmäßig auf 1 gesetzt, da die Fakultät von 0 laut Definition 1 ist. In der `while`-Schleife findet die Berechnung statt (falls n != 0).

Die Funktion `range()`

(1) **range()**

Zum Überprüfen schreiben wir auf der Python Shell bspw. `list(range(5))`. Im Code (d. h. wenn wir ein Programm schreiben) benötigen wir für die Ausgabe nach wie vor zusätzlich noch die print-Funktion, also z. B. `print(list(range(5)))`.

		Merke:
a) `range(5)`:	`0, 1, 2, 3, 4`	→ Standard-Beginn ist 0
b) `range(-1, 2)`:	`-1, 0, 1`	
c) `range(17, 17)`:	keiner/leer	→ 2. Parameter ist exklusive
d) `range(17, 18)`:	`17`	
e) `range(6, 3)`:	keiner/leer	→ Standard-Schrittweite ist 1
f) `range(6, 3, -1)`:	`6, 5, 4`	→ negative Schrittweite möglich
g) `range(0, 4, 2)`:	`0, 2`	→ größere Schrittweite möglich
h) `range(-4, 6, 3)`:	`-4, -1, 2, 5`	
i) `range(4, 6, 0.5)`:	Fehler	→ Schrittweite ganzzahlig (≠ 0)

(2) [*Zusatzaufgabe*] **Sequenz gesucht!**

Die Parameter der Funktion **range(**[start,] **stop**[, step]**)** können wie folgt gewählt werden, damit die Zahlen -3, 1 und 5 (und **nur diese** Zahlen) enthalten sind:

- `range(-3, 6, 4)`
- `range(-3, 7, 4)`
- `range(-3, 8, 4)`
- `range(-3, 9, 4)`
- `range(5, -4, -4)`
- `range(5, -5, -4)`
- `range(5, -6, -4)`
- `range(5, -7, -4)`

Sequenzen können also sowohl aufsteigend, als auch absteigend angeordnet sein, je nachdem ob der Parameter `step` positiv oder negativ gewählt wird. `step` darf außerdem nicht 0 sein (logischerweise)! Wichtig ist, dass `range(-3, 5, 4)` ist **nicht möglich** ist, nachdem der Parameter `stop` exklusive ist und 5 somit nicht mehr in der `range` liegen würde.

`for`-Schleifen

(1) Iterieren und Quadrieren

Lösungsvorschlag:

```
n = 7   # von 1 bis n

for i in range(1, n+1, 2):
    print(i**2)
```

`for i in range(...)` bedeutet, dass der nachfolgende Teil (*for*-Block) für alle Elemente `i` aus der Zahlensequenz `range(...)` ausgeführt wird. `range(1, n+1, 2)` repräsentiert eine Sequenz von Zahlen von `1` (inklusive) bis `n+1` (exklusive). Die `2` gibt die Schrittweite an (normal: 1). `range(1, n, 2)` ist nicht ausreichend, da in diesem Fall `n` selbst nicht inbegriffen wäre, `n` aber laut Aufgabenstellung ebenfalls behandelt werden soll.

(2) [*Zusatzaufgabe*] `in`-Operator bei Strings

Der `in`-Operator ist der einzige Zugehörigkeitsoperator („*membership operator*“) in Python. Wie die Vergleichsoperatoren und die booleschen Operatoren wertet er zu einem **booleschen Wert** (*bool*) aus. Er liefert *True*, falls das Element bzw. die Zeichenkette in der `range` bzw. dem String (*str*) rechts des Operators vorhanden ist, anderenfalls *False*.

Beispiele:

True: `"el" in "Hello World"` oder `1 in range(5)`
False: `"A" in "abcdefg"` oder `2 in range(1, 7, 2)`

Achtung: Im Konstrukt der *for*-Schleife wird der Operator dazu verwendet, um über alle Elemente zu iterieren, die in der Sequenz (`range`) liegen. In diesem Fall wertet `i in range(...)` also **nicht** zu einem booleschen Wert aus, sondern weist der Variable `i` in jedem Durchlauf (d. h. in jeder „Iteration“) den „nächsten“ Wert der Sequenz zu.

(3) `i` hat im `else`-Teil den zuletzt behandelten Wert, nicht etwa den nächsten Wert! Angenommen unser Beispiel aus Aufgabe (2) mit `n = 7` würde noch über einen `else`-Teil verfügen: Die `for`-Schleife würde alle Werte von 1 bis 7 (also hier 1, 3, 5 und 7) behandeln und `i` hätte im `else`-Teil bzw. nach der Schleife den Wert 7, nicht etwa 8.

Fortsetzung auf der nächsten Seite!

Für besonders Interessierte (Zusatzinformationen):

Ein bisschen knifflig wird es beim Verlassen von verschachtelten Schleifen (sog. *„nested loops“*) mit `break`. `break` verlässt immer nur die *innerste* Schleife (`while`/`for`), kann sich aber nicht auf eine *äußere* Schleife beziehen. Manchmal möchte man allerdings zwei verschachtelte Schleifen verlassen.
Hierzu gibt es Tricks wie bspw.:

```
for i in range(1, 5):       # äußere Schleife (über i)
    for j in range(1, 4):   # innere Schleife (über j)
        print(i, j, i*j)
        if i*j == 8:        # bei i = 4, j = 2:
            break           #  verlasse innere Schleife
    else:
        continue            # fahre in äußerer Schleife fort
    break                   # verlasse äußere Schleife
```