12. Iterationen

Problem: Eine bestimmte Anzahl von Primzahlen (Liste von Primzahlen) soll multipliziert werden. Die Primzahlen sind in einer also in einer Liste gespeichert.

In [43]:
```
primzahlen = [2,3,5,7,11]

product = primzahlen[0]* primzahlen[1]*primzahlen[2]*primzahlen[3]*primzahle
print(product)
```

```
2310
```

⇒ Dieser Code ist schreibaufwendig und unübersichtlich

⇒ Umständlich, bei großen Listen

andere Lösung: Interationen

12.1 For-Schleife

In [44]:
```
primzahlen = [2,3,5,7,11]
product = 1
for number in primzahlen:
    product = product * number #same as product *= number

print (product)
```

```
2310
```

Syntax

```
for var in expresssion:
        block
```

- Schlüsselworte sind `for` und `in`
- 1. Zeile heißt Schleifenkopf
- 2. Zeile Schleifenrumpf `block` mit einer oder mehreren Anweisungen
- Schleifenvariable `var` im Schleifenkopf
- Schleifeniteration ist ein Durchlauf (Ausführnung eines Schleifenrumpfs)

Beispiel:

In [1]:
```
for ch in "Hallo":
    print(ch * 2)
```

```
HH
aa
ll
ll
oo
```

In [3]:
```
for ingredient in ("sugar", "water", "oil"):
    if ingredient == "sugar":
        print("sweet!")
```

```
sweet!
```

Ablauf der Schleife beeinflussen

- `break` - Schleifenrumpf wird umehend beendet (eventuell vorzeitg)
- `continue` - die aktuelle Schleifeniteration wird vorzeitig beendet, es wird in den Schleifenkopf gesprungen und die Schleifenvariable wird auf den nächsten Wert gesetzt.
- `else`– Zweig - wird nach Beendigungder Schleife ausgeführt, genau dann wenn die Schleife nicht mit `break` verlassen wurde.

Beispiel

In [45]:
```
noten_und_anzahl =[("sehr gut", 1), ("gut", 3), ("befriedigend",0), ("ausrei

for na in noten_und_anzahl:
    note , anzahl = na
    if anzahl ==0:
        continue
    if note == "sehr gut":
        print("Tolle Leistung")
        break
else:
    print("Durchschnitt war in Ordnung")
```

```
Tolle Leistung
```

Alternativ kann man das auch so aufschreiben:

In [4]:
```
noten_und_anzahl =[("sehr gut", 1), ("gut", 3), ("befriedigend",0), ("ausrei

for note, anzahl in noten_und_anzahl:
    if anzahl ==0:
        continue
    if note == "sehr gut":
        print("Tolle Leistung")
        break
else:
    print("Durchschnitt war in Ordnung")
```

```
Tolle Leistung
```

Nützliche Funktionen

- `range`
- `zip`
- `reversed`

12.2 `range`

- erzeugt eine Folge von Indices für die Schleifendurchläufe, genauer einen Iterator.
- `range(stop)` - ergibt 0, 1, ..., stop -1
- `range(start, stop)` - ergibt start, start + 1, ... , stop -1
- `range(start, stop, steo)` - ergibt start, start + 2* step, ... , stop -1

Beispiele:

In [46]:
```
range(5)
```

Out[46]:
```
range(0, 5)
```

In [6]:
```
print(range(2, 30, 5)) #keine Liste, print gibt keine Ausgabe
```

```
range(2, 30, 5)
```

In [5]:
```
list(range(0,9))
```

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

In [48]:
```
list (range(2, 30, 5))
```

Out[48]:
```
[2, 7, 12, 17, 22, 27]
```

In [49]:
```
for i in range(3,6):
    print (i, "**3 = ", i ** 3)
```

```
3 **3 =  27
4 **3 =  64
5 **3 =  125
```

12.3 `zip`

- Funktion
- nimmt eine oder mehrere Sequenzen und liefert eine "Liste" von Tupeln mit korrespndierenden Elementen
- erzeugt eigentlich keine Liste, sondern einen Iterator

Beipsiel

In [50]:
```
fleisch = ["Rind", "Huhn", "Hund"]
beilage =["Kartoffeln", "Pommes", "Spätzle"]
```

```
print(list(zip(fleisch, beilage)))
```

```
[('Rind', 'Kartoffeln'), ('Huhn', 'Pommes'), ('Hund', 'Spätzle')]
```

- mit `zip` können mehrere Sequenzen parallel durchlaufen werden
- bei unterschiedlicher Länge ist das Ergbnis so lang, wie die kürzere Sequenz.

Beispiel

In [51]:
```
for xyz in zip("rind", "huhn", range(5,10)):
    x, y, z = xyz
    print(x, y, z)
```

```
r h 5
i u 6
n h 7
d n 8
```

12.4 `reversed`

- Funktion
- Durchlauf einer Sequenz in umgekehrter Richtung

Beispiel

In [52]:
```
for x in reversed("hallo"):
    print(x)
```

```
o
l
l
a
h
```

12.5 Beispiel: Implemtierung einer Fakultätsfunktion

Fakultät ist wie folgt definiert:

$$0! = 1$$
$$(n+1)! = (n+1) \cdot n!$$

Aufgabe: Entwickle eine Funktion `fact`, welche die Fakulät einer ganzen Zahl berechnet.

1. Bezeichner und Datentypen

- Die Eingabe ist `n: int` (mit n >= 0)
- Die Ausgabe ist `int`

2. Funktionsgerüst

```
def fact(
    n: int # assume n>=0
    )
    # fill in
    return
```

3. Beispiele

```
assert fact(0) == 1
assert fact(1) == 1
assert fact(3) == 6
```

4. Ergebnis

In [26]:
```
def fact(
        n: int
        ) -> int:
    result = 1
    for i in range (1, n+1):
        result= result * i
    return result
```

In [31]:
```
print(fact(10))
```

```
3628800
```