# 10_Uebungsaufgaben_Loesungen

## 0.1 Vorschläge für Übungsaufgaben

Die folgenden Übungsaufgaben sind ausführlicher als die unterwegs vorgeschlagenen Mini-Übungen. Das Ziel ist es hier, nicht nur einzelne Themen direkt auszuprobieren, sondern vielmehr verschiedene Themen zu kombinieren, um so ein verknüpfendes Verständnis der Lerninhalte zu erreichen.

### 0.1.1 Comprehensions

Es sei $\mathbb{N} = \{1, 2, \dots\}$, d.h. die natürlichen Zahlen ohne die Null. Legen Sie die folgenden Objekte in Python an:

- Liste $[x_0, x_1, \dots, x_{n-1}]$ mit $x_i := a + i\Delta x$ für $i = 0, 1, \dots, n$, $\Delta x := \frac{b-a}{n}$ für ein gegebenes Intervall $[a, b] \subset \mathbb{R}$ und $n \in \mathbb{N}$
- Menge $\{(a, b) \in \mathbb{N}^2 \,|\, a, b \in \mathbb{N}, \frac{a}{b} \in \mathbb{N}\backslash\{1\}, a \leq 10, b \leq 10\}$, d.h. die Menge sollte zum Beispiel das Elemente $(2, 1)$ enthalten, nicht aber $(2, 2)$ oder $(2, 3)$
- Menge $\{(a, b, c, d) \in \mathbb{N}^4 | \, a^2 + b^2 + c^2 = d^2, a \leq 10, b \leq 10, c \leq 10, d \leq 10\}$

**Hinweis** Für die zweite Teilaufgabe muss überprüft werden, ob $\frac{a}{b}$ eine natürliche Zahl ist.

[2]:
```
# Erstes Beispiel

a = 1
b = 4
n = 10

deltax = (b-a)/n
result = [ a+i*deltax for i in range(0,n+1) ]
print(result)
```

```
[1.0, 1.3, 1.6, 1.9, 2.2, 2.5, 2.8, 3.1, 3.4, 3.6999999999999997, 4.0]
```

[27]:
```
# Zweites Beispiel

import math

result = { (a,b) for a in range(1,11)
                 for b in range(1,11)
                 if math.fabs(a/b-int(a/b)) < 1e-8
                 and math.fabs(a/b-1) > 1e-8 }
```

```
# Die Comprehension sollte klar sein. Wichtig ist der Test auf Ganzzahligkeit
# und der zur Exklusion der 1, bei denen Gleitkomma-Vergleiche notwending sind.

for zaehler,nenner in result:
    print("{:d}/{:d}={:.2f}".format(zaehler,nenner,zaehler/nenner))
```

```
6/2=3.00
10/5=2.00
7/1=7.00
2/1=2.00
8/4=2.00
8/1=8.00
3/1=3.00
6/1=6.00
9/1=9.00
9/3=3.00
10/1=10.00
5/1=5.00
4/2=2.00
8/2=4.00
6/3=2.00
4/1=4.00
10/2=5.00
```

[14]:
```
# Drittes Beispiel

import math

result = { (a,b,c,d) for a in range(1,11)
                     for b in range(1,11)
                     for c in range(1,11)
                     for d in range(1,11)
                     if a**2+b**2+c**2==d**2 }
#print(result)
for q in result:
    print("{}: {:d} + {:d} + {:d} = {:d}".
 ↪format(q,q[0]**2,q[1]**2,q[2]**2,q[3]**2))
```

```
(4, 2, 4, 6): 16 + 4 + 16 = 36
(2, 4, 4, 6): 4 + 16 + 16 = 36
(1, 2, 2, 3): 1 + 4 + 4 = 9
(4, 4, 2, 6): 16 + 16 + 4 = 36
(8, 1, 4, 9): 64 + 1 + 16 = 81
(2, 1, 2, 3): 4 + 1 + 4 = 9
(3, 6, 6, 9): 9 + 36 + 36 = 81
(1, 4, 8, 9): 1 + 16 + 64 = 81
(4, 7, 4, 9): 16 + 49 + 16 = 81
```

```
(6, 3, 6, 9): 36 + 9 + 36 = 81
(4, 8, 1, 9): 16 + 64 + 1 = 81
(3, 2, 6, 7): 9 + 4 + 36 = 49
(7, 4, 4, 9): 49 + 16 + 16 = 81
(4, 1, 8, 9): 16 + 1 + 64 = 81
(6, 2, 3, 7): 36 + 4 + 9 = 49
(2, 2, 1, 3): 4 + 4 + 1 = 9
(2, 3, 6, 7): 4 + 9 + 36 = 49
(8, 4, 1, 9): 64 + 16 + 1 = 81
(3, 6, 2, 7): 9 + 36 + 4 = 49
(6, 6, 3, 9): 36 + 36 + 9 = 81
(1, 8, 4, 9): 1 + 64 + 16 = 81
(2, 6, 3, 7): 4 + 36 + 9 = 49
(6, 3, 2, 7): 36 + 9 + 4 = 49
(4, 4, 7, 9): 16 + 16 + 49 = 81
```

### 0.1.2 Pascalsches Dreieck

Das Pascalsche Dreieck hat interessante Eigenschaften in der Kombinatorik und steht in direktem Zusammenhang mit dem Binomialkoeffizienten. Seine $n$-te Stufe ist wie folgt definiert:

- Für $n = 1$ ist die Stufe als pascal(n) = [1] definiert.
- Für $n = 2$ ist die Stufe als pascal(n) = [1, 1] definiert.
- Für $n > 2$ ist die Stufe als [1, p[0]+p[1], p[1]+p[2], ..., p[n-3]+p[n-2], 1] definiert, wobei die Zahlen p durch p = pascal(n-1) gegeben sind.

Implementieren Sie eine Funktion pascal(n), die die $n$-te Stufe zurückgibt. Beispiel: Für $n = 10$ sollte die Ausgabe [1, 9, 36, 84, 126, 126, 84, 36, 9, 1] erscheinen, und für $n = 3$ die Ausgabe [1, 2, 1].

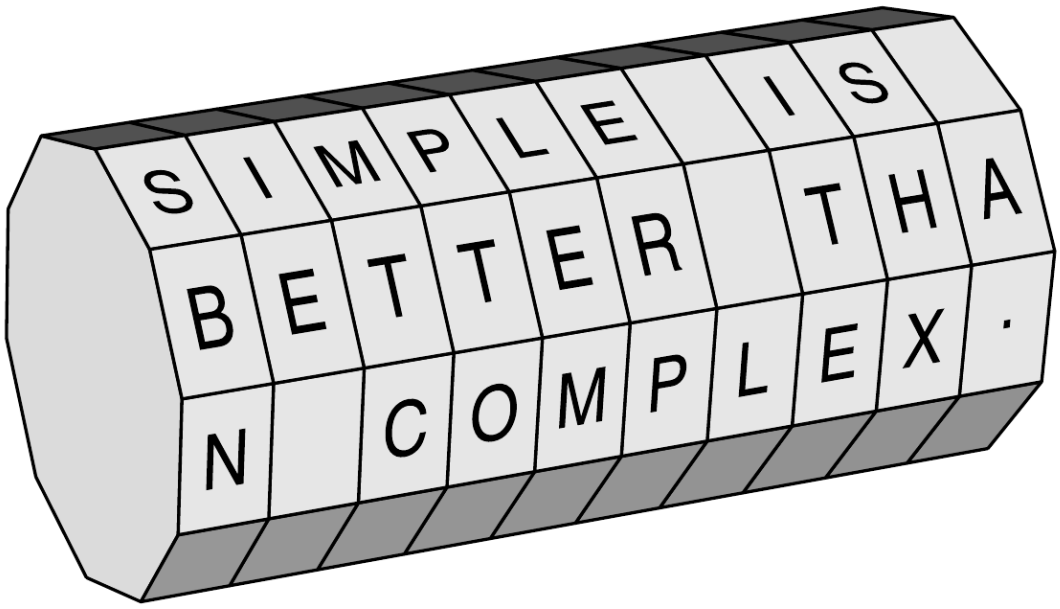
[26]:
```python
def pascal(n):
    if n==1:
        return [1]
    if n==2:
        return [1,1]
    else:
        # Zuerst: alle benötigten p der Stufe davor,
        # das spart viele viele viele rekursive Aufrufe
        p = pascal(n-1)
        # Dann: berechnen der neuen Stufe
        result = [ p[i] + p[i+1] for i in range(0,n-2) ]
        # Schliesslich: manuelle Ergänzung der Einsen am Anfang und am Ende
        result.insert(0,1)
        result.append(1)
        return result

print(pascal(3))
print(pascal(10))
```

```
[1, 2, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
```

### 0.1.3 Skytale Verschlüsselung

Diese Verschlüsselungstechnik geht zurück auf die Spartaner, ca. 700 v.Chr. Bei einer Skytale handelt es sich um einen Stab mit bestimmten Durchmesser, auf den ein langer Papierstreifen aufgewickelt wird und die Botschaft geschrieben wird, beispielsweise so wie im Bild dargestellt.




Wickelt man den Papierstreifen wieder ab, erhält man für das obige Bespiel (length=10, circ=3 – sinngemäß eine dreieckige Skytale) einen Streifen mit der Beschriftung

SBNIE MTCPTOLEMERP LITESHX A.

und für das obige Bespiel (length=10, circ=4 – sinngemäß eine viereckige Skytale) einen Streifen mit der Beschriftung

SBN IE MTC PTO LEM ERP L ITE SHX A.

Mit einer Skytale vom gleichen Durchmesser kann die Botschaft dann wieder entschlüsselt werden.

Schreiben Sie eine Funktion skytale_encode(text, length, circ) , in welche die Verschlüsselung einer Botschaft mithilfe einer Skytale realisiert wird. Dabei ist `text` die Botschaft. Die Argumente length und circ sollen die Länge sowie der Umfang der Skytale, jeweils in Anzahl an Textzeichen, sein. Überprüfen Sie dabei zuerst ob die Botschaft kurz genug ist, d.h. auf eine Skytale mit entsprechender Länge und Umfang passt.

Schreiben Sie eine weitere Funktion skytale_decode(text, length, circ), in welcher die Entschlüsselung mithilfe einer Skytale realisiert wird. Vermeiden Sie Code-Duplizierung, indem Sie die Funktion skytale_encode wiederverwenden.

Eine schöne Beschreibung für das nächste Projekt mit Kindern im Grundschulalter aus dem Bekannten- und Verwandtenkreis findet sich hier.

Abbildung: Maria Alkämper, IANS, Uni Stuttgart, CC-BY

[29]:
```
def skytale_encode(text,length,circ):
    message = ""
    len_text = len(text)
    # Berechnung der maximalen Textlänge der Skytale
    len_skytale = length*circ

    if (len_text>len_skytale):
        print("Text does not fit on the skytale.")
        return message

    # Ergänze den Text um so viele Leerzeichen am Ende, dass die
    # Skytale komplett gefüllt ist
    text = text + " "*(len_skytale-len_text)

    # Verschlüsselung ist nun einfach: wir nehmen einfach
    # jeden n-ten Buchstaben, bis wir alle erwischt haben,
    # hierbei ist n= die Länge der Skytale
    for i in range(0,length):
        message = message + text[i::length]
    return message

# Entschlüsselung ist einfach: Tausch von Länge und Umfang
def skytale_decode(text,length,circ):
    return skytale_encode(text,circ,length)

# Test
text = "SIMPLE IS BETTER THAN COMPLEX."
length = 10
circ = 4

message = skytale_encode(text, length, circ)
print(message)
text = skytale_decode(message, length, circ)
print(text)
```

```
SBN IE  MTC PTO LEM ERP   L ITE SHX  A.
SIMPLE IS BETTER THAN COMPLEX.
```

## 0.2 Impressum

### 0.2.1 Programmierkurs Python, Dominik Göddeke https://www.ians.uni-stuttgart.de, Universität Stuttgart

Version vom April 2023

[ ]: