Serie 5

Aufgabe 1:

Schreiben Sie eine Funktion `primes(n)`, die alle Primzahlen, die kleiner gleich $n \in \mathbb{N}$ sind, als Liste zurückgibt. Testen Sie Ihre Funktion mit verschiedenen Eingabewerten.

Aufgabe 2:

Schreiben Sie eine Funktion `pfz(n)`, die die *Primfaktorzerlegung* der natürlichen Zahl n berechnet, und als Dictionary der Form `{"Primzahl": Exponent}` zurückgibt. Die Primfaktorzerlegung einer natürlichen Zahl ist definiert als $n = p_1^{e_1} \cdots p_r^{e_r}$, für Primzahlen $p_1 < \ldots < p_r \leq n$ und Exponenten $e_1 \ldots, e_r \in \mathbb{N}$. Falls ein Exponent e_i gleich Null ist, soll diese Primzahl nicht im Dictionary vorkommen. Sie dürfen Ihre Funktion aus Aufgabe 1 verwenden. Falls Sie Aufgabe 1 nicht gelöst haben, dürfen Sie sich auf $n \leq 20$ beschränken (Die Primzahlen kleiner 20 sind: 2, 3, 5, 7, 11, 13, 17, 19).

Aufgabe 3:

Schreiben Sie eine Funktion `encrypt(m)` die ein Wort mithilfe der *Caesar-Verschlüsselung* verschlüsselt und zurückgibt und eine Funktion `decrypt(m)`, die für ein verschlüsseltes Wort wieder den Klartext zurückgibt. Hinweis: Die Caesar-Verschlüsselung, verschiebt jeden Buchstaben um 1 Position im Alphabet. `"A"` $\rightarrow$ `"B"` usw. Tipp: Nutzen Sie dafür das folgende Dictionary.

In [ ]:
```
letters = {"A":1,"B":2,"C":3,"D":4,"E":5,"F":6,
           "G":7,"H":8,"I":9,"J":10,"K":11,"L":12,
           "M": 13, "N": 14, "O": 15, "P": 16,"Q": 17,
           "R": 18,"S": 19,"T": 20,
           "U":21, "V":22, "W": 23, "X": 24,
           "Y": 25, "Z": 26}
```

Aufgabe 4:

Knacken Sie den Tresor, indem Sie alle Möglichkeiten der 4-stelligen Kombination durchprobieren (also die Funktion `tresor` mit allen Kombinationen aufrufen). Die Kombinationen bestehen aus den Ziffern 0-9 und müssen als String übergeben werden (z.B. "1234"). Sie haben den richtigen Code gefunden, sobald "Tresor geöffnet! Code ist: ..." ausgegeben wird.

In [ ]:
```
from hashlib import sha256
def tresor(code):
    secret_code = ("b2b2f104d32c638903e151a9b20d6e27"
        + "b41d8c0c84cf8458738f83ca2f1dd744")
```

```
    m = sha256()
    m.update(code.encode())
    if m.hexdigest() == secret_code:
        print("Tresor geöffnet! Code ist:", code)
```

Falls Sie daran interessiert sind, was in der Funktion `tresor` passiert: Die Variable `secret_code` enthält einen sogenannten Hash-Wert des geheimen Codes. Ein Hash-Wert ist eine Art Fingerabdruck eines Textes, der es erlaubt, den Text zu überprüfen, ohne ihn direkt zu speichern. Es ist (nach derzeitigem Wissensstand) unmöglich, vom Hash-Wert direkt auf den Code zurückzuschließen. Wenn der Hash-Wert des eingegebenen Codes mit dem gespeicherten Hash-Wert übereinstimmt, dann gilt mit sehr hoher Wahrscheinlichkeit, dass der eingegebene Code korrekt ist. Die Wahrscheinlichkeit, dass zwei verschiedene Codes denselben Hash-Wert erzeugen, ist 1 zu $2^{256} \approx 10^{77}$ (Die Anzahl der Atome im Universum wird auf 10^{80} geschätzt).

Aufgabe 5:

Schreiben Sie eine Funktion, die eine Tic-Tac-Toe Position als geschachtelte Liste übernimmt, und überprüft ob ein Spieler gewonnen hat. Wenn ja, soll `X gewinnt` oder `O gewinnt` zurückgegeben werden. Ein möglicher Spielstand könnte so aussehen:

In [ ]:
```
position = [["X","O","X"],
            ["X","X","O"],
            ["O","X","O"]]
```

Aufgabe 6:

Collatz-Vermutung: Man beginne mit einer beliebigen natürlichen Zahl $n \in \mathbb{N}$ und setzt $c_0 = n$. Die weitere Folge wird für $i \geq 0$ definiert durch:

$$c_{i+1} = \begin{cases} \frac{c_i}{2}, & \text{falls } c_i \text{ gerade ist} \\ 3c_i + 1, & \text{falls } c_i \text{ ungerade ist} \end{cases}$$

Die Kollatz-Vermutung besagt, dass es für jede natürliche Zahl n eine Zahl $k \in \mathbb{N}$ gibt, so dass $c_k = 1$ ist.

Schreiben Sie eine Funktion `collatz(n,m)`, welche die ersten m Elemente der Collatz-Folge $c_0, \ldots c_{m-1}$ für eine gegebene natürliche Zahl n berechnet und als Liste zurückgibt. Der zweite Rückgabewert soll die Anzahl der Schritte sein, die benötigt werden, bis die Folge zum ersten Mal den Wert 1 erreicht (also das kleinste k mit $c_k = 1$). Falls es keine solche Zahl k in den ersten m Elementen gibt, soll `None` zurückgegeben werden. Testen Sie Ihre Funktion mit verschiedenen Eingabewerten.