Aufgabe 1

Beschreibe den Begriff *Rekursion* in der Programmierung.

Aufgabe 2

Beschreibe je einen Vor- und einen Nachteil des Programmierens mit einer Rekursion.

Aufgabe 3

Welche Ausgabe macht die folgende Python-Funktion?

```
def f(n):
    if n == 1:
        return 7
    else:
        return n + f(n-1)

print(f(4))
```

Aufgabe 4

Welche Ausgabe macht die folgende Python-Funktion?

```
def f(n):
    if n < 2:
        return 3
    else:
        return 2*f(n-1) + 1

print(f(4))
```

Aufgabe 5

Welche Ausgabe macht die folgende Python-Funktion?

```
def f(n):
    if n < 3:
        return n
    elif n % 2 == 0:
        return 2*f(n-1)
    else:
        return 2+f(n-1)

print(f(5))
```

Aufgabe 6

Berechne die Werte der unten abgebildeten rekursiven Funktion $f(n)$ für $n = 1, 2, 4, 5$. Schreibe danach eine Funktion $g(n)$, die mit einer `for`-Schleife und ohne Rekursion die gleichen Werte wie $f(n)$ berechnet und als Werte zurückgibt.

Funktionen, welche eine Aufgabe mittels Schleifen und ohne Rekursion lösen, werden *iterativ* genannt.

```
def f(n):
    if n == 1:
        return n
    else:
        return n + f(n-1)
```

Aufgabe 7

Definiere eine Python-Funktion `factorial(n)`, welche die Werte $n!$ für $n = 0, 1, 2, \ldots$ rekursiv berechnet.

Aufgabe 8

Definiere eine Python-Funktion `factorial(n)`, welche die Werte $n!$ für $n = 0, 1, 2, \ldots$ iterativ, also mit einer Schleife berechnet. Verwende zum Platzsparen eine Einrückung von 2 statt 4 Zeichen.

Aufgabe 9

Definiere eine Python-Funktion `fib(n)`, welche die Werte

`fib(1)=1, fib(2)=1, fib(3)=2, fib(4)=3, fib(5)=5, fib(6)=8, ...`

rekursiv berechnet. Verwende eine Einrückungstiefe von 2 statt 4 Zeichen.

Aufgabe 10

Warum verlangsamt sich die Ausführung der rekursiven Funktion zur Berechnung der Werte der Fibonacci-Folge immer stärker?

Aufgabe 11

Definiere eine Python-Funktion `fib(n)`, welche die Werte

`fib(1)=1, fib(2)=1, fib(3)=2, fib(4)=3, fib(5)=5, fib(6)=8, ...`

iterativ berechnet. Verwende eine Einrückungstiefe von 2 statt 4 Zeichen.