Hier sehen die das Grundschema, mittels dem man aus vorhandenen Funktionen g, h eine neue primitiv-rekursive Funktion f definieren kann:

```
1 def f(t,x):
2   temp = g(x)
3   for i in range(t):
4     temp = h(temp, i, x)
5   return temp
```

Problem 1. Demonstrieren Sie anhand der Funktion $t!$ (`factorial`), wie man dieses Schema einsetzt.

Problem 2. Erinnern Sie sich an die Fibonacci-Reihe

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

Wie können Sie die Funktion $n \mapsto F_n$ primitiv-rekursiv implementieren? Was ist hierbei die Schwierigkeit?

Problem 3. Ich definiere jetzt etwas ähnliches wie die Fibonacci-Reihe:

$$G_n := \begin{cases} n & \text{if } n = 0, 1, 2 \\ G_{n-1} + G_{n-3} & \text{else.} \end{cases}$$

Können Sie das primitiv-rekursiv berechnen?

Problem 4. Ich mache es jetzt noch schwieriger:

$$H_n := \begin{cases} 1 & \text{if } n = 0 \\ H_{n-1} + H_{\lfloor n/2 \rfloor} & \text{else.} \end{cases}$$

Wie geht das? Geht das mit dem selben Trick? Mit einem anderen Trick?

Problem 5. Erinnern Sie sich an die Péter-Ackermann-Funktion $A_m(n)$:

$$A(m,n) := \begin{cases} n+1 & \text{if } m = 0, \\ A(m-1,1) & \text{if } m \geq 1, n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m, n \geq 1 \ . \end{cases}$$

Können Sie skizzieren, wie man diese Funktion ohne Rekursion mithilfe einer `while`-Schleife berechnen kann, beispielsweise in Python?

Problem 6. Skizzieren Sie eine Turing-Maschine, die die Ackermann-Funktion berechnet. Verwenden Sie so viele Bänder, wie Sie wollen. Wie codieren Sie die Eingabe (m, n)?

Problem 7. Im Kapitel über Boolesche Schaltkreise haben wir die Majority-Funktion $\mathrm{Maj}_n(x_1, \ldots, x_n)$ definiert. Erkären Sie, wie Sie einen Schaltkreis für Maj_n bauen würden. Sie dürfen gerne mit einem "schlechten" Schaltkreis anfangen.