Zusätzliche Aufgaben zur Rekursion

Taxonomie der Aufgaben:

* E: Einstieg, für alle Pflicht
* Ü: Weitere Übungsaufgabe.
* V: Vertiefung. Das bestehende Wissen wird vertieft und es gibt neue Erkenntnisse zu gewinnen
* H: Herausforderung. Die Aufgabe ist anspruchsvoll aber deswegen auch besonders interessant.

# Aufgabe 1 (Ü)

Versuchen Sie folgendes Programm zu verstehen. Was machen die einzelnen Definitionen? Am besten testen Sie dies an verschiedenen Werten für a und b aus. Benennen Sie anschliessend die Funktionen um, so das deutlich wird, was diese machen und ersetzen Sie ‘bla’ in der letzten Anweisung durch einen passenderen Begriff.

def foo(n,m):

return foo2(n,m,0)

def foo2(n,m,a):

if n<m:

return a

else:

return foo2(n-m,m,a+1)

def bar(n,m):

if n<m:

return n

else:

return bar(n-m, m)

a=67

b=7

print(foo(a,b),'bla',bar(a,b))

# Aufgabe 2 (V)

Es lässt sich zeigen, dass man um alle mathematisch berechenbaren Funktionen berechnen zu können, gar nicht so viele Grundbausteine braucht. Beispielsweise lassen sich die Addition von beliebigen Zahlen durch den rekursiven Aufruf der Addition und Subtraktion mit 1 durchführen. (Die Addition mit 1 bezeichnet man auch als Nachfolgerfunktion und die Subtraktion mit 1 entsprechend als Vorgängerunktion).

1. Definieren Sie in dieser Weise eine rekursiv arbeitende Funktion für die Addition.
2. Definieren Sie in dieser Weise eine rekursiv arbeitende Funktion für die Subtraktion.

# Aufgabe 3 (H)

Sie haben nun gesehen, dass Sie das Potenzieren auf die Multiplikation, die Multiplikation auf die Addition und die Addition auf die Berechnung von Vorgänger und Nachfolger zurückführen können. Aufgabe: Definieren Sie die Funktionen vorgaenger(n) und nachfolger(n) und nutzen Sie diese, um mittels Rekursionen die Funktion potenziere(n,a) zu erstellen. Dabei dürfen nur die beiden Funktionen vorgaenger(n) und nachfolger(n) sowie darauf aufbauende Hilfsfunktionen genutzt werden.

Die Funktionen vorgaenger(n) und nachfolger(n) brauchen nicht rekursiv definiert zu werden.

# Aufgabe 4 (H)

(Idee aus dem Mathe-Känguru). Eine Prima-Primzahl ist wie folgt definiert:

* Jede einstellige Primzahl ist eine Prima-Primzahl
* Eine mehrstellige Primzahl ist genau dann eine Prima-Primzahl, wenn beim Weglassen der ersten oder letzten Ziffer eine Prima-Primzahl übrigbleibt.

Schreiben Sie eine Funktion is\_prima\_primzahl(n), die für eine Zahl n prüft, ob Sie eine Prima-Primzahl ist. Schreiben Sie anschliessend eine Funktionen prima\_primzahlen(), die Ihnen alle maximal Prima-Primzahlen in der Shell ausgibt.

Erstellen Sie eine iterative und eine rekursive Variante, welche die Prima-Primzahlen berechnet. Beschränken Sie Sich bei der iterativen Variante auf maximal 4-stellige Zahlen.

Warum ist es hier nicht sinnvoll, die beiden Programmiervarianten hinsichtlich der Laufzeiten zu vergleichen, bzw. warum wird sich hier kaum ein Unterschied finden?

# Aufgabe 5 (H)

Informieren Sie Sich über das Spiel «Die Türme von Hanoi».

Implementieren Sie eine rekursiv arbeitende Funktion, die Ihnen sagt, in welcher Reihenfolge Sie die Scheiben zwischen den Türmen bewegen sollen. Sie dürfen Sich gerne auch Inspirationen aus dem Internet holen. Die Aufgabe gilt als erledigt, wenn Ihre Implementierung funktioniert und Sie diese erklären können.

Zusatz: Testen Sie die Funktion und ergänzen Sie diese so, dass Sie die Anzahl Züge zählen können. Das funktioniert am besten mit einer globalen Variablen. Informieren Sie Sich, was das ist und wie Sie diese innerhalb von Python nutzen können.