LU03b - Rekursion

Die Rekursion ist eine Technik in der Programmierung, bei der eine Funktion sich selbst aufruft, um ein Problem zu lösen. Rekursive Funktionen bestehen normalerweise aus zwei Teilen: dem Basisfall, der das Ende der Rekursion markiert, und dem rekursiven Fall, der die Funktion erneut mit einem Teilproblem aufruft. Rekursion kann für viele Probleme eine elegante und effektive Lösung bieten.

Rekursion muss mit Sorgfalt verwendet werden, da sie ohne einen klaren Basisfall zu einer Endlosschleife führen kann.

Animation: Call Stack und Aufrufbaum

Rekursion ist schwer zu lesen, weil mehrere Aufrufe derselben Funktion gleichzeitig offen sind. Die Animation macht das sichtbar:

- **Tab 1** zeigt am Beispiel `factorial(3)`, wie sich vier Frames stapeln, bis der Basisfall erreicht ist – und wie die eigentliche Rechnung erst auf dem Rückweg passiert.
- **Tab 2** zeigt am Beispiel `fibonacci(4)`, dass hier kein Stapel, sondern ein **Baum** entsteht – und dass dabei dieselben Teilprobleme mehrfach berechnet werden.

Die Animation verwendet kleinere Zahlen als die Beispiele unten (`factorial(3)` statt `factorial(5)`, `fibonacci(4)` statt `fibonacci(6)`), damit Stack und Baum vollständig auf den Bildschirm passen. Das Prinzip ist identisch.

Beispiel 1: Fakultät

Die Fakultät einer Zahl ist das Produkt aller ganzen Zahlen von 1 bis zu dieser Zahl. Sie wird oft mit dem Symbol ! dargestellt, z.B. $5!=5\times4\times3\times2\times1=120$.

```
def factorial(n):
    if n == 0:  # Basisfall
        return 1
    else:       # Rekursiver Fall
        return n * factorial(n-1)
if __name__ == '__main__':
    print(factorial(5)) # Ausgabe: 120
```

Beispiel 2: Fibonacci-Reihe

Die Fibonacci-Reihe ist eine Folge von Zahlen, bei der jede Zahl die Summe der beiden vorhergehenden ist. Die ersten zwei Zahlen in der Fibonacci-Reihe sind 0 und 1.

```python
def fibonacci(n):
    if n == 0:  # Basisfall 1
        return 0
    elif n == 1:  # Basisfall 2
        return 1
    else:       # Rekursiver Fall
        return fibonacci(n-1) + fibonacci(n-2)
if __name__ == '__main__':
    print(fibonacci(6)) # Ausgabe: 8
```

Vorsicht bei der Zählung: Die Folge startet bei `n = 0`, nicht bei `n = 1`.

n	0	1	2	3	4	5	6
fibonacci	0	1	1	2	3	5	8

`fibonacci(6)` ist also **8** – die 5 gehört zu `fibonacci(5)`. Solche Verwechslungen sind bei Reihen, die bei 0 beginnen, die häufigste Fehlerquelle.

Rekursion hat ihren Preis: Der Aufrufbaum von `fibonacci(4)` enthält bereits 9 Aufrufe, obwohl es nur 5 verschiedene Teilprobleme gibt – `fibonacci(1)` wird dreimal komplett neu berechnet. Bei `fibonacci(30)` sind es über 2,7 Millionen Aufrufe. Wer das vermeiden will, speichert Zwischenergebnisse (Memoization). Tab 2 der Animation oben zeigt die Mehrfachberechnungen farblich markiert.

Rekursion ist eine mächtige Technik, die in vielen verschiedenen Problembereichen eingesetzt werden kann. Durch das Verständnis des Basisfalls und des rekursiven Falls können Sie rekursive Funktionen erstellen, um eine Vielzahl von Problemen auf eine klare und elegante Weise zu lösen.

M323-LU03, M323-C1B