Rekursion

Definition: Eine rekursive Prozedur, Funktion oder Methode besitzt mehrere Abbruchpfade, wobei sie sich in mindestens einem Abbruchpfad selbst aufruft. Das zugrunde liegende Programmierkonzept der Rekursion ist dabei das Zurückführen einer Aufgabe auf eine einfachere Teilaufgabe der selben Klasse. Jeder Funktionsaufruf löst jeweils nur ein Teil des Gesamtproblems bis letztendlich ein Abbruchpfad ohne rekursiven Selbstaufruf erreicht wird. Für jede rekursive Methode gibt es eine äquivalente iterative Lösung.

Durch die Rekursion kann der Programmierstil, in Bezug auf Lesbarkeit und Schwierigkeit der Implementierung, in einigen Fällen deutlich verbessert werden. Die Mehrzahl aller Funktionen sind jedoch rein iterativ und Selbstaufrufe stellen die Ausnahme dar.

Vorteile	**Nachteile**
Kürzere Formulierung	Schlechteres Laufzeitverhalten
Leichter verständliche Loesung	(Overhead von Funktionsaufrufen)
Einsparung von Variablen	
Teils sehr effiziente Problemloesungen	

Im Folgenden sind 2 Beispiele für rekursive Anwendungsfälle aufgeführt, wobei jeweils die rekursive und iterative Implementierung gegenüber gestellt ist.

Beispiel 1 - Berechnung der Fakultät:
Fakultät von der Zahl n ist:

$$n \cdot (n-1) \cdot (n-2) \cdot (n-3) ... \cdot 2 \cdot 1$$

Iterative Berechnung:

```
1 static int fakultaet(int n) {
2   int ergebnis = 1;
3   for (int i = 1; i <= n; i++) {
4     ergebnis = ergebnis * i;
5   }
6   return ergebnis;
7 }
```

Rekursive Berechnung:

```
1 static int fakultaet(int n) {
2   if (n == 1) {
3     return 1;
4   } else {
5     return fakultaet(n-1) * n;
6   }
7 }
```

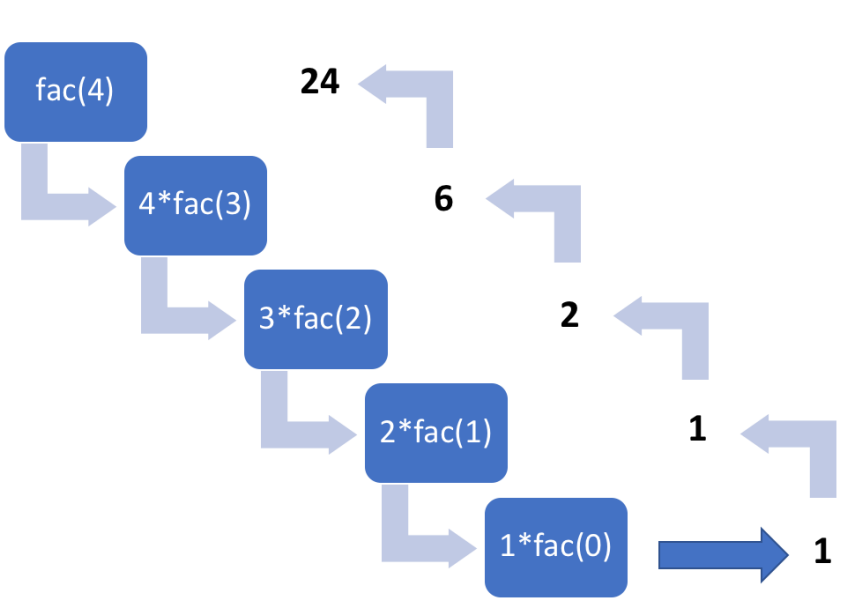

Abbildung 1: Rekursiver Funktionsaufruf am Beispiel der Fakultät

Beispiel 2 - Berechnung der Fibonacci-Folge:
Ein weiterer typischer Anwendungsfall der Rekursion bietet die Fibonacci-Zahlenfolge. Ein beliebtes Beispiel für die Fibonacci-Zahlenfolge ist dabei die Fortpflanzungsfolge von Kaninchen, welche wie folgt beschrieben werden kann:

- Die Folge beginnt mit einem Paar junger Kaninchen
- Nach einem Monat ist ein Kaninchen fortpflanzungsfähig
- Ein Kaninchenpaar (im Fortpflanzungsalter) gebärt jeden Monat ein weiteres Kaninchenpaar

Die Fibonacci-Folge: 1, 1, 2, 3, 5, 8, 13, 21, ...

$$Fib(i) = Fib(i-2) + Fib(i-1), i > 1$$

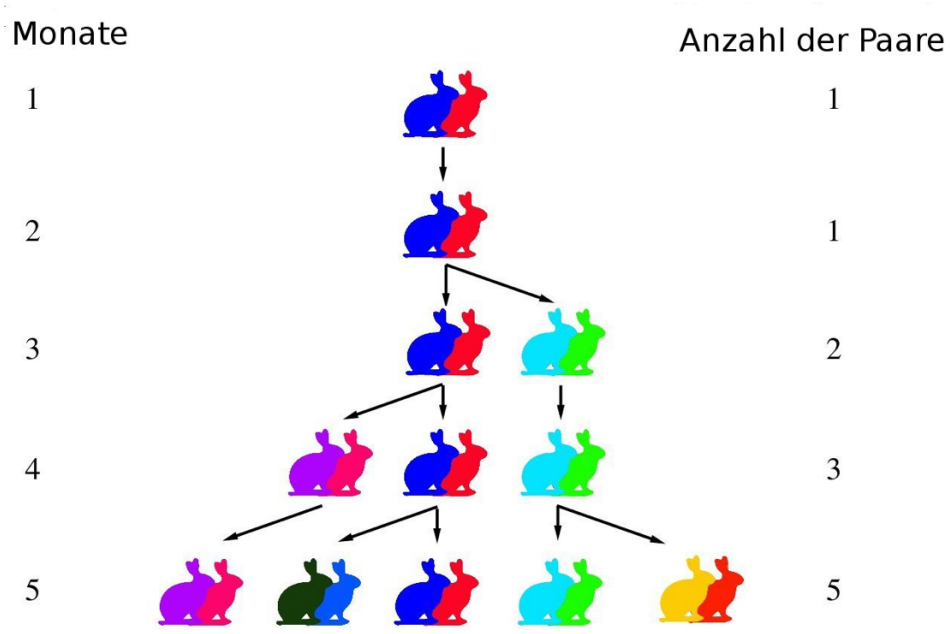

Abbildung 2: Fortpflanzungsfolge von Kaninchen als Fibonacci-Folge
Source: http://math.unifr.ch/plantexpo/pdf/03_fibonnaciDE.pdf

Iterative Berechnung:

```
1  static int fibonacci (int n) {
2    if (n <= 0) {
3      return 0;
4    } else if (n == 1) {
5      return 1;
6    } else {
7      int a = 0; // hat am Anfang der Schleife den Wert Fib(i-2)
8      int b = 1; // hat am Anfang der Schleife den Wert Fib(i-1)
9      int i = 2;
10     while (i <= n) {
11       int aa = b; // Wert von Fib(i-1)
12       int bb=a+b; // Wert von Fib(i)
13       a = aa; // Hilfsvariable f{"u}r Folgedurchgang
14       b = bb; // Hilfsvariable f{"u}r Folgedurchgang
15       i++;
16     }
17     return b;
18   }
19 }
```

Rekursive Berechnung:

```
1  public static int fibonacci(int n) {
2    if (n <= 0) {
3      return 0;
4    } else if (n == 1) {
5      return 1;
6    } else {
7      return fibonacci(n - 2) + fibonacci(n - 1);
8    }
9  }
```

Während beim Beispiel der Berechnung der Fakultät die anfänglich genannten Vorteile nicht ganz so offensichtlich sind, so zeigt die Fibonacci-Zahlenfolge doch eine deutlich verringerte Zeilenanzahl und leichter verständliche Lösung.

Beispiele:

- Fakultät: `https://codeocean.openhpi.de/exercises/284/implement`
- Fibonacci: `https://codeocean.openhpi.de/exercises/285/implement`

Links zur weiteren Vertiefung:

- Wikipedia: Rekursion
- Moritz Lenz / Perlgeek: Was ist Rekursion
- Hochschule RheinMain: Vergleich iterativ vs. rekursiv an Beispielen