1. Rekursion

Rekursion bezeichnet die Lösung eines Algorithmischen Problems durch eine Funktion, die sich selbst aufruft.

Oft lassen sich komplizierte Probleme rekursiv sehr leicht lösen. Ein Nachteil liegt in der hohen Speicherbelastung rekursiver Funktionen. Eine Funktion ruft sich oft mehrmals hintereinander selbst auf, wobei jeweils eine Kopie des Funktion im Speicher des Rechners gehalten wird.

1. Beispiel

Bsp.: Berechne die Fakultät einer Zahl, z.B. von 5.
Die Fakulatät von 5 (auch kurz als 5! geschrieben) wird berechnet als 5 * 4 * 3 * 2 * 1 = 120. Dabei gilt 0! = 1.

Die rekursive Lösung funktioniert dann oft sehr gut, wenn eine Funktion gelöst werden kann, indem der Operand leicht modifiziert nochmals aufgerufen wird, solange bis ein Aufruf mit einem Operanden erfolgt der einfach durch Angabe des Ergebnisses gelöst werden kann.

z.B. 1! = 1 Diese einfache Lösung soll die Funktion kennen.

Für jede Zahl größer als 1 kann die Lösung berechnet werden, indem der Operand mal der Fakultät einer Zahl multipliziert wird, die um 1 verringert wurde:

```
5! =
5 * 4! =
5 * 4 * 3! =
5 * 4 * 3 * 2! =
5 * 4 * 3 * 2 * 1! =
5 * 4 * 3 * 2 * 1  = 120 oder allgemein

$i! =
$i * ($i-1)! =
$i * ($i-1) * ($i-2)! =
$i * ($i-1) * ($i-2) * ($i-3)! = ... usw.
```

Dies lässt sich nun in PHP leicht implementieren:

```
function fak($i){

  if ($i==1)
     return 1;
  else
     return $i * fak($i-1);
// Die Funktion ruft sich hier also selbst (rekursiv) auf!
}
```

Ein Aufruf der Funktion fakultaet(5) führt dann zu folgenden Aufrufen:

```
Erster Aufruf:          return 5 * fakultaet(4);
```

Bevor das Ergebnis dieser Funktion mit return zurückgeliefert wird, muss erst noch fakultaet(4) berechnet werden:

```
Zweiter Aufruf:         return (4 * fakultaet(3));
Dritter Aufruf:         return (3 * fakultaet(2));
Vierter Aufruf:         return (2 * fakultaet(1));
Fünfter Aufruf:         return (1);
```

Erst jetzt kann der fünfte Aufruf der Funktion ein Ergebnis liefern. Es sind jetzt aber auch fünf Funktionen im Speicher. Die Aufrufe 1 bis 4 konnten nicht beendet werden, sie warten auf das Ergebnis der nachfolgenden Aufrufe.

Der fünfte Aufruf meldet dem vierten nun das Ergebnis 1;
Der vierte Aufruf kann dann das Ergebnis return (2 *1) als Ergebnis an den dritten Aufruf zurückliefern usw.

Die Funktionen sind also gleichsam in den Keller hinabgestiegen, um das einzige Problem zu suchen, das zum Ende führen kann, nämlich fakultaet(1). Jeder rekursiver Aufbruch benötigt eine Abbruchbedingung, wann der ständige Selbstaufruf zu Ende ist.

Der Rekrusive Aufruf muss uns dieser Lösung einen Schritt näher bringen, hier indem der Operand jeweils um 1 verringert wird.

```
                                                    120
Erster Aufruf:          return (5 * fakultaet(4));
                                                    24
Zweiter Aufruf:         return (4 * fakultaet(3));
                                                    6
Dritter Aufruf:         return (3 * fakultaet(2));
                                                    2
Vierter Aufruf:         return (2 * fakultaet(1));
                                                    1
Fünfter Aufruf:         return (1);
```