

Der Call-Stack und die Rekursion

Ein populäres Beispiel für rekursive Algorithmen ist die Fakultätsfunktion:

```
5! = 5*4*3*2*1
fakultaet(5) = 120
fakultaet(3) = 3*2*1 = 6
```



(A1) Iterativ

Implementiere in BlueJ eine iterative Version der Fakultätsfunktion, die als Argument die Zahl entgegennimmt, deren Fakultät berechnet werden soll.



(A2) Rekursiv

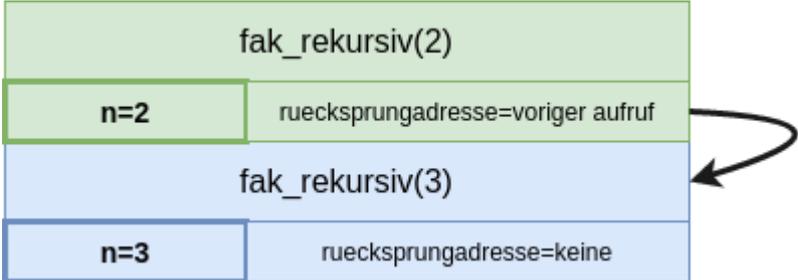
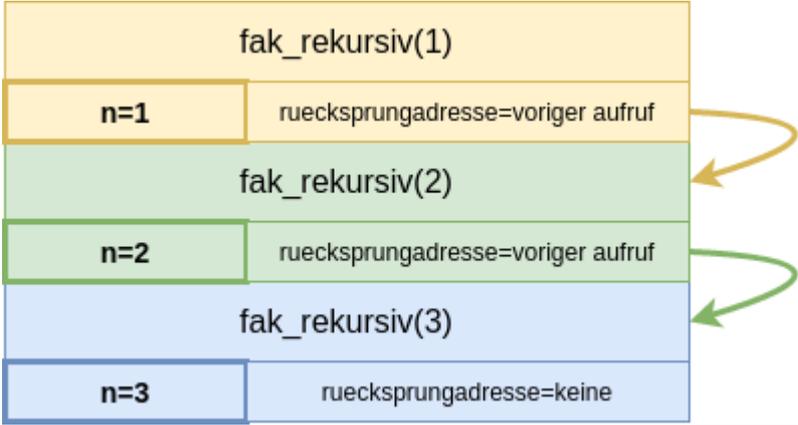
Implementiere anhand des folgenden Pseudocodes eine rekursive Version fak_rekursiv.

```
fak_rekursiv(int n):
  wenn n=1 oder n=0:
    return 1
  sonst:
    return n*fak_rekursiv(n-1)
```

- Was ist der Rekursionsfall, was der Basisfall?
- Teste deine rekursive Methode

Detaillierte Betrachtung des Call-Stacks bei der Rekursion

Was passiert	Wie sieht der Stack aus?

Was passiert	Wie sieht der Stack aus?
<p>fak_rekursiv(3) wird aufgerufen. Auf dem Stack wird Speicher für diesen Aufruf reserviert. Es gibt keine Rücksprungadresse. Innerhalb dieses Aufrufs wird fak_rekursiv(2) (nächster Schritt) aufgerufen, da die Fallunterscheidung nicht zum Basisfall führt sondern zum Rekursionsfall.</p>	 <p>The diagram shows a single stack frame for the function <code>fak_rekursiv(3)</code>. The frame is divided into two sections: the top section contains <code>n=3</code> and the bottom section contains <code>rücksprungadresse=keine</code>.</p>
<p>fak_rekursiv(2) wird aus dem vorhergehenden Aufruf heraus aufgerufen. Auf dem Stack wird Speicher für diesen Aufruf reserviert: Wichtig: Jeder Aufruf hat seinen eigenen Speicherbereich für Variablen, d.h. jeder Aufruf von fak_rekursiv hat sein eigenes n auf die die anderen Aufrufe nicht zugreifen können. Die Rücksprungadresse befindet sich jetzt im vorigen Aufruf der rekursiven Methode selbst. Das ist anders, als beim ersten Beispiel für den Call-Stack! Da erneut der Rekursionsfall eintritt, wird aus diesem Aufruf heraus fak_rekursiv(1) aufgerufen.</p>	 <p>The diagram shows two stack frames. The bottom frame is for <code>fak_rekursiv(3)</code> with <code>n=3</code> and <code>rücksprungadresse=keine</code>. The top frame is for <code>fak_rekursiv(2)</code> with <code>n=2</code> and <code>rücksprungadresse=voriger aufruf</code>. A curved arrow points from the <code>rücksprungadresse</code> field of the <code>fak_rekursiv(2)</code> frame back to the <code>fak_rekursiv(3)</code> frame.</p>
<p>fak_rekursiv(1) wird aus dem vorhergehenden Aufruf heraus aufgerufen. Auf dem Stack wird Speicher für diesen Aufruf reserviert. Die Rücksprungadresse befindet sich wiederum im vorigen Aufruf der rekursiven Methode selbst. Jetzt tritt der Basisfall ein!</p>	 <p>The diagram shows three stack frames. From bottom to top: <code>fak_rekursiv(3)</code> with <code>n=3</code> and <code>rücksprungadresse=keine</code>; <code>fak_rekursiv(2)</code> with <code>n=2</code> and <code>rücksprungadresse=voriger aufruf</code>; and <code>fak_rekursiv(1)</code> with <code>n=1</code> and <code>rücksprungadresse=voriger aufruf</code>. Curved arrows point from the <code>rücksprungadresse</code> field of each frame back to the frame immediately below it.</p>

Was passiert	Wie sieht der Stack aus?
<p>Dies ist der erste Aufruf, der vollständig abgeschlossen ist. Es wird kein neuer Aufruf von fak_rekursiv auf den Call-Stack gelegt, sondern der Aufruf von fak_rekursiv(1) endet damit, dass 1 an die aufrufende Stelle zurückgegeben wird und die zum Aufruf gehörenden Daten vom Stack entfernt werden.</p>	<p>The diagram shows a call stack with three frames. The top frame, <code>fak_rekursiv(1)</code>, is crossed out with a large red 'X'. Below it are two active frames: <code>fak_rekursiv(2)</code> (green) and <code>fak_rekursiv(3)</code> (blue). Each frame has a sub-section for <code>n</code> and <code>ruecksprungadresse</code>. For <code>fak_rekursiv(1)</code>, <code>n=1</code> and <code>ruecksprungadresse=voriger aufruf</code>. For <code>fak_rekursiv(2)</code>, <code>n=2</code> and <code>ruecksprungadresse=voriger aufruf</code>. For <code>fak_rekursiv(3)</code>, <code>n=3</code> and <code>ruecksprungadresse=keine</code>. A yellow box labeled <code>return 1</code> has an arrow pointing to the return address field of the crossed-out <code>fak_rekursiv(1)</code> frame.</p>
<p>Damit hat der Aufruf von fak_rekursiv(2) alle Informationen, um abgeschlossen zu werden: Der Ausdruck $n \cdot \text{fak_rekursiv}(n-1)$ kann jetzt zu $2 \cdot 1$ ausgewertet und an die aufrufende Stelle zurückgegeben werden. Die zum Aufruf gehörenden Daten werden vom Stack entfernt.</p>	<p>The diagram shows a call stack with two frames. The top frame, <code>fak_rekursiv(2)</code>, is crossed out with a large red 'X'. Below it is one active frame: <code>fak_rekursiv(3)</code> (blue). Each frame has a sub-section for <code>n</code> and <code>ruecksprungadresse</code>. For <code>fak_rekursiv(2)</code>, <code>n=2</code> and <code>ruecksprungadresse=voriger aufruf</code>. For <code>fak_rekursiv(3)</code>, <code>n=3</code> and <code>ruecksprungadresse=keine</code>. A green box labeled <code>return 2*1</code> has an arrow pointing to the return address field of the crossed-out <code>fak_rekursiv(2)</code> frame.</p>
<p>Jetzt hat auch der Aufruf von fak_rekursiv(3) alle Informationen, um abgeschlossen zu werden: Der Ausdruck $n \cdot \text{fak_rekursiv}(n-2)$ kann jetzt zu $3 \cdot 2$ ausgewertet und an die aufrufende Stelle zurückgegeben werden. Die zum Aufruf gehörenden Daten werden vom Stack entfernt. Der Call-Stack ist leer, der Aufruf der Methode beendet.</p>	<p>The diagram shows a call stack with one frame. The frame, <code>fak_rekursiv(3)</code>, is crossed out with a large red 'X'. Each frame has a sub-section for <code>n</code> and <code>ruecksprungadresse</code>. For <code>fak_rekursiv(3)</code>, <code>n=3</code> and <code>ruecksprungadresse=keine</code>. A blue box labeled <code>return 3*2</code> has an arrow pointing to the return address field of the crossed-out <code>fak_rekursiv(3)</code> frame.</p>

Zusammenfassung

- **Rekursion** bedeutet, dass eine Funktion/Methode sich selbst aufruft.
- Alle rekursiven Funktionen haben eine Fallunterscheidung: den **Basisfall** und den **Rekursionsfall**.

- Die Funktionsaufrufe werden auf dem Aufruf-Stack gespeichert, dabei wird dieser immer größer bis der Basisfall eintritt. Anschließend wird der Stack von oben nach unten "abgearbeitet", bis er leer ist und damit der Aufruf der rekursiven Methode endet.
- Der Aufruf-Stack kann unter Umständen sehr groß werden und sehr viel Arbeitsspeicher belegen.

From:
<https://info-bw.de/> -

Permanent link:
https://info-bw.de/faecher:informatik:oberstufe:algorithmen:rekursion:callstack_rekursion:start

Last update: **11.12.2023 15:51**

