

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

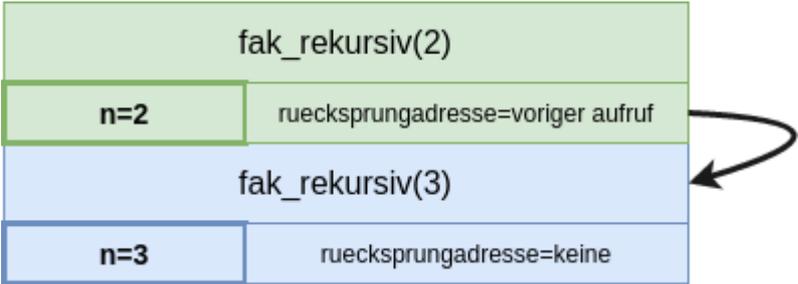
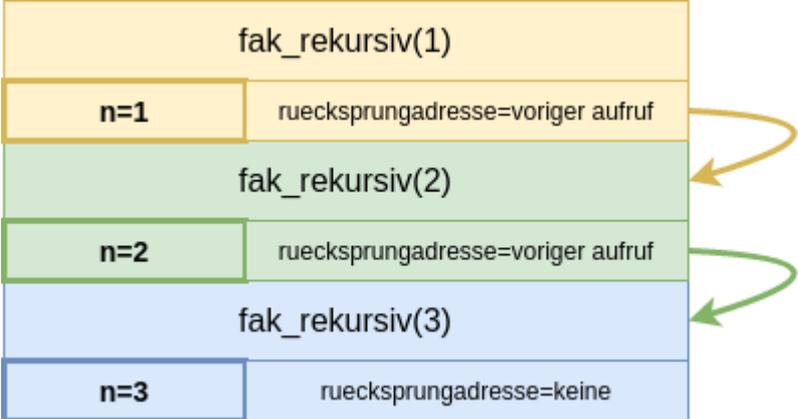
Implementiere anhand des folgenden Pseudocodes eine rekursive Version fak_rekursiv.

```
fak_rekursiv(int n):
  wenn n=1:
    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 a light blue rectangle divided into two parts: a smaller box on the left containing <code>n=3</code> and a larger box on the right containing <code>ruecksprungadresse=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 <code>fak_rekursiv(3)</code> (light blue) with <code>n=3</code> and <code>ruecksprungadresse=keine</code>. The top frame is <code>fak_rekursiv(2)</code> (light green) with <code>n=2</code> and <code>ruecksprungadresse=voriger aufruf</code>. A curved arrow points from the return address 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> (light blue) with <code>n=3</code> and <code>ruecksprungadresse=keine</code>; <code>fak_rekursiv(2)</code> (light green) with <code>n=2</code> and <code>ruecksprungadresse=voriger aufruf</code>; and <code>fak_rekursiv(1)</code> (light yellow) with <code>n=1</code> and <code>ruecksprungadresse=voriger aufruf</code>. Curved arrows point from the return address fields of the top two frames back to their respective callers.</p>

Was passiert	Wie sieht der Stack aus?
<p>Jetzt ist der erste Aufruf vollständig abgeschlossen. 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>	

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

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:algorithmen:rekursion:callstack_rekursion:start?rev=1642074255

Last update: **13.01.2022 11:44**

