Programmeerimine

13. loeng

Täna loengus

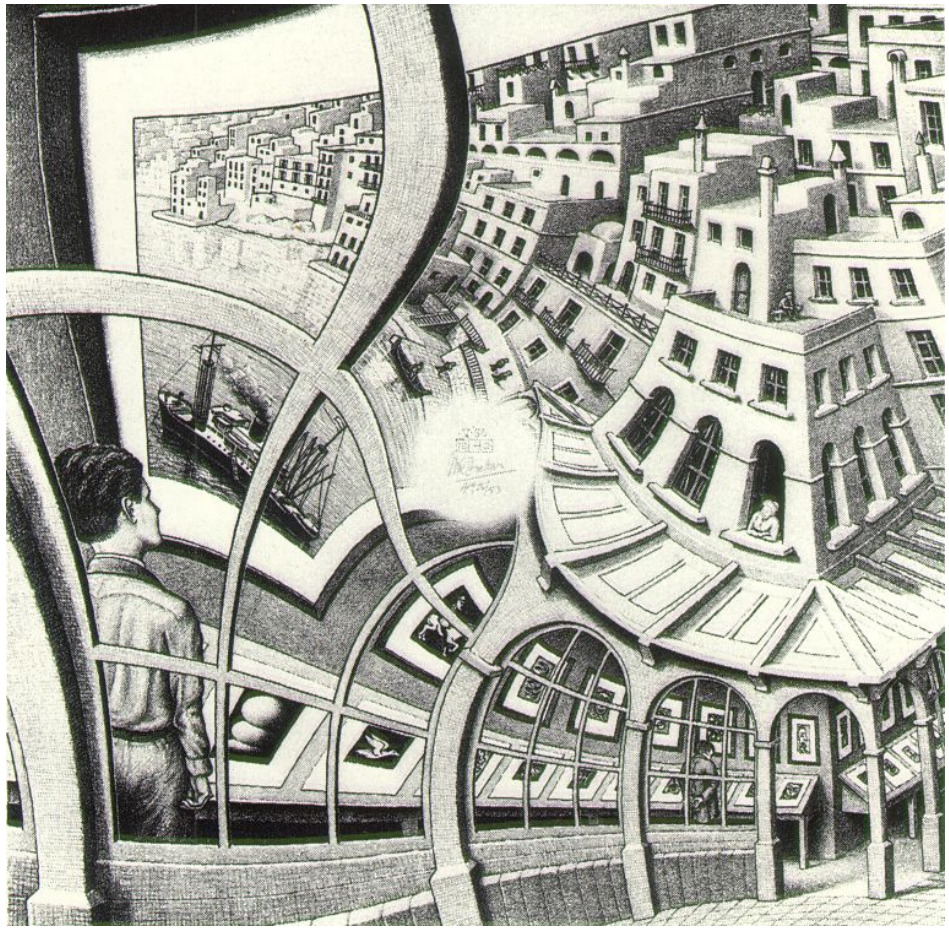
- Rekursiooni mõiste
- Rekursiooni baas ja samm
- Rekursiivsete funktsioonide näiteid

Rekursioon

- Rekursioon on funktsioonide defineerimise meetod, kus defineeritav funktsioon kutsub välja iseennast.
- Sarnaselt iteratsioonile (so. tsüklidirektiividele) võimaldab rekursioon kirjeldada korduvtäidetavaid protsesse.
- Üldjuhul on rekursioon võimsam kui iteratsioon.
  - Tsüklidirektiive saab modelleerida rekursiivsete funktsioonidena.
  - Rekursiooni saab modelleerida tsüklidirektiividega üldjuhul ainult *magasini* (ehk *pinu*) abil.
- Rekursiooni kasutatakse laialdaselt programmeerimises, arvutiteaduses, matemaatikas, aga ka näiteks keeleteaduses, muusikas, kunstis, ...

M. C. Escher "Pildigalerii" (1956)

- Leideni Ülikooli töörühm Hendrik Lenstra juhtimisel täiendas Escheri joonistust "Pildigalerii" täites keskel oleva tühimiku.
- `http://escherdroste.math.leidenuniv.nl/`



Faktoriaal

- Naturaalarvu n faktoriaal (tähistus $n!$) on n esimese positiivse täisarvu korrutis.

$$n! = \prod_{i=1}^{n} i = n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

- On kokku lepitud, et $0! = 1$.

Iteratiivne faktoriaal

```
def factorial(n):
    f = 1
    for i in range(1,n+1):
        f *= i
    return f
```

Faktoriaal

- Paneme tähele, et

$$\begin{aligned}(n+1)! &= (n+1)\cdot n\cdot (n-1)\cdot \ldots \cdot 3\cdot 2\cdot 1\\ &= (n+1)\cdot n!\end{aligned}$$

- Seega saame rekursiivse definitsiooni

$$n! = \begin{cases} 1, & \text{kui } n = 0\\ n\cdot (n-1)!, & \text{kui } n > 0\end{cases}$$

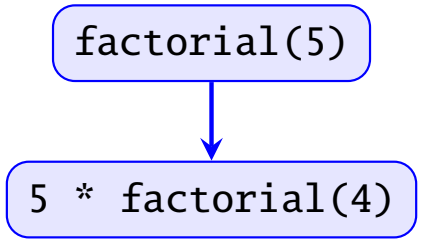
Rekursiivne faktoriaal

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

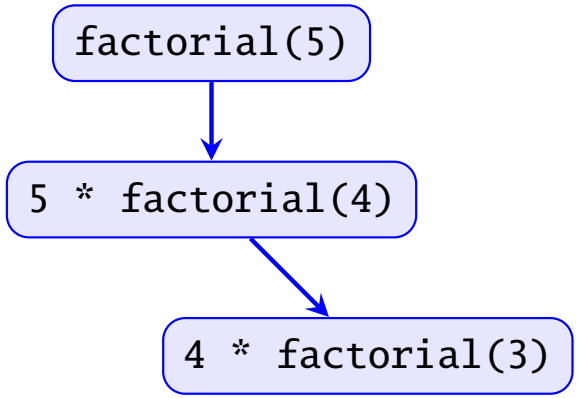
Faktoriaal

```
factorial(5)
```

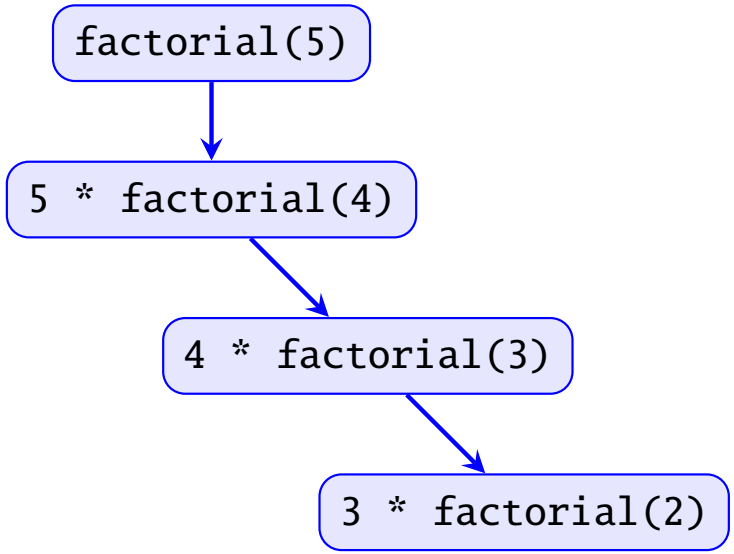
Faktoriaal



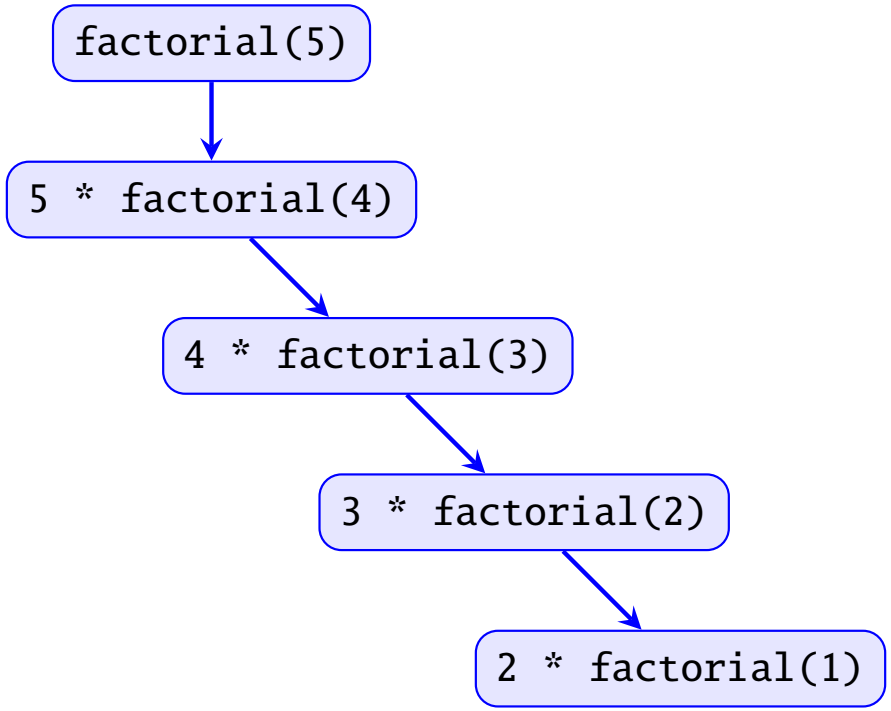
Faktoriaal



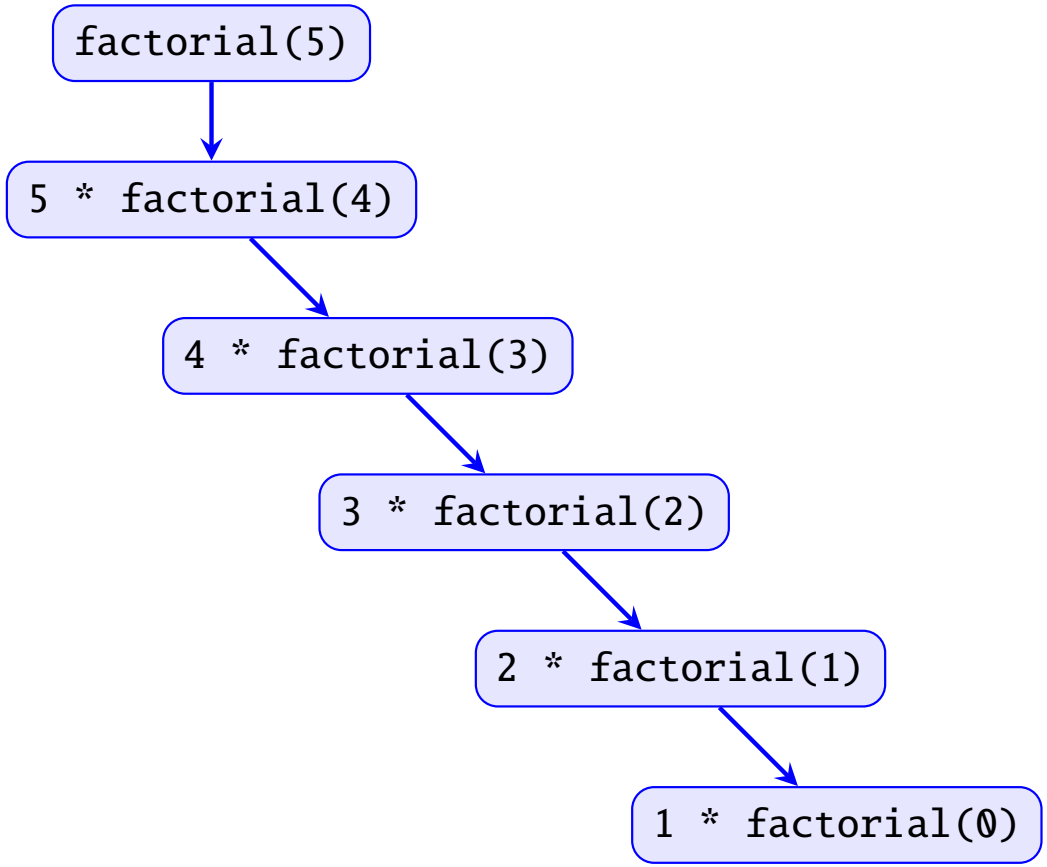
Faktoriaal



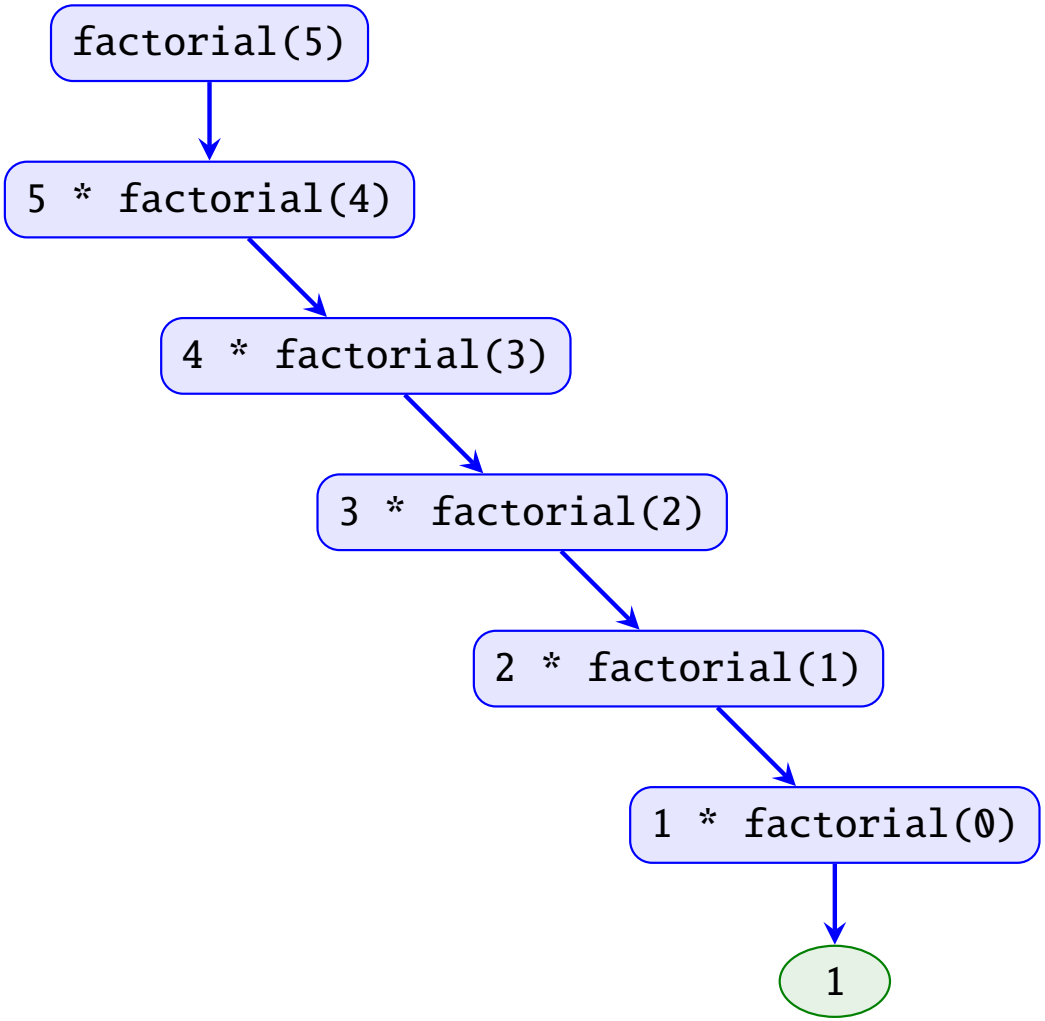
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)

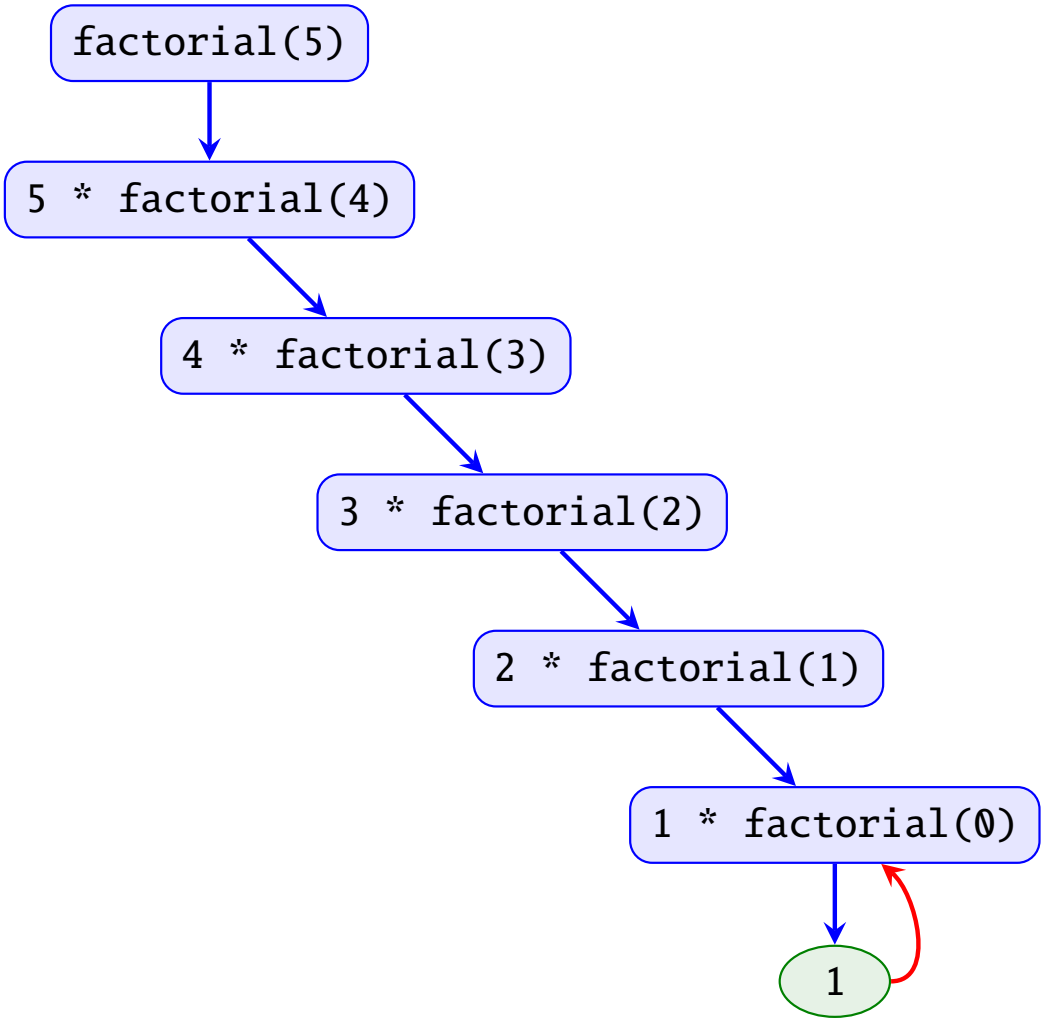
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1 * factorial(0)

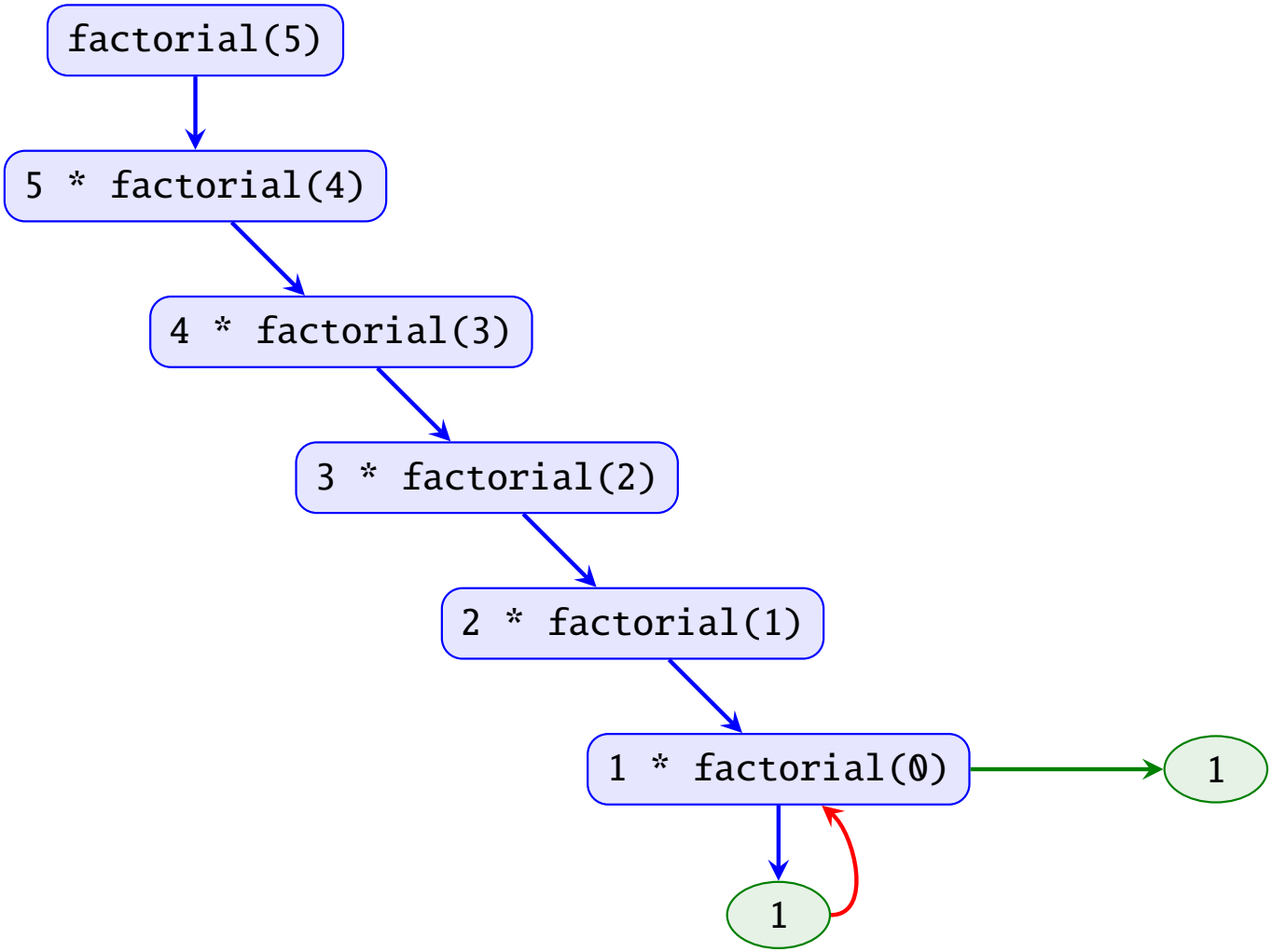
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1 * factorial(0)
1

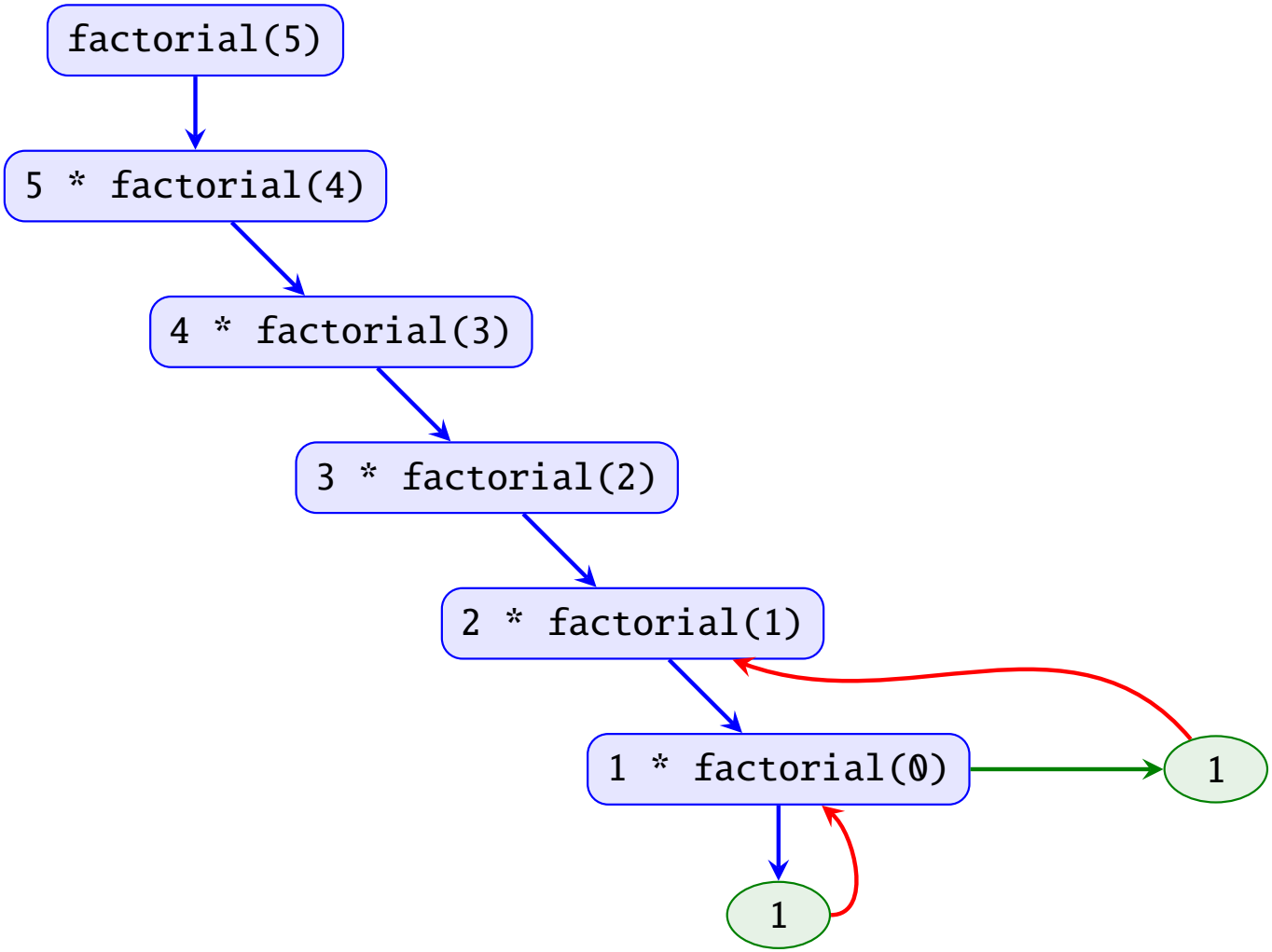
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1 * factorial(0)
1

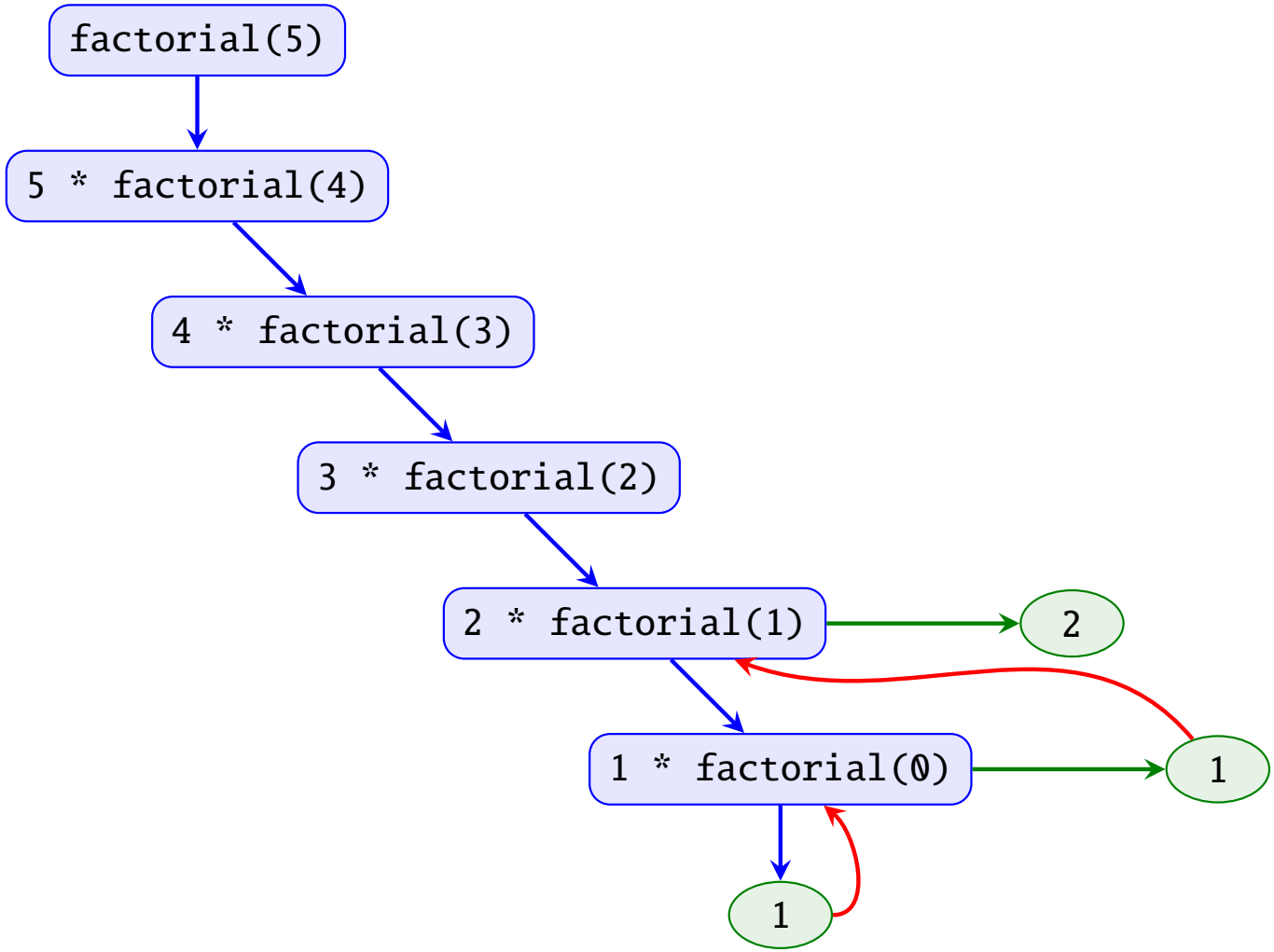
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1 * factorial(0)
1
1

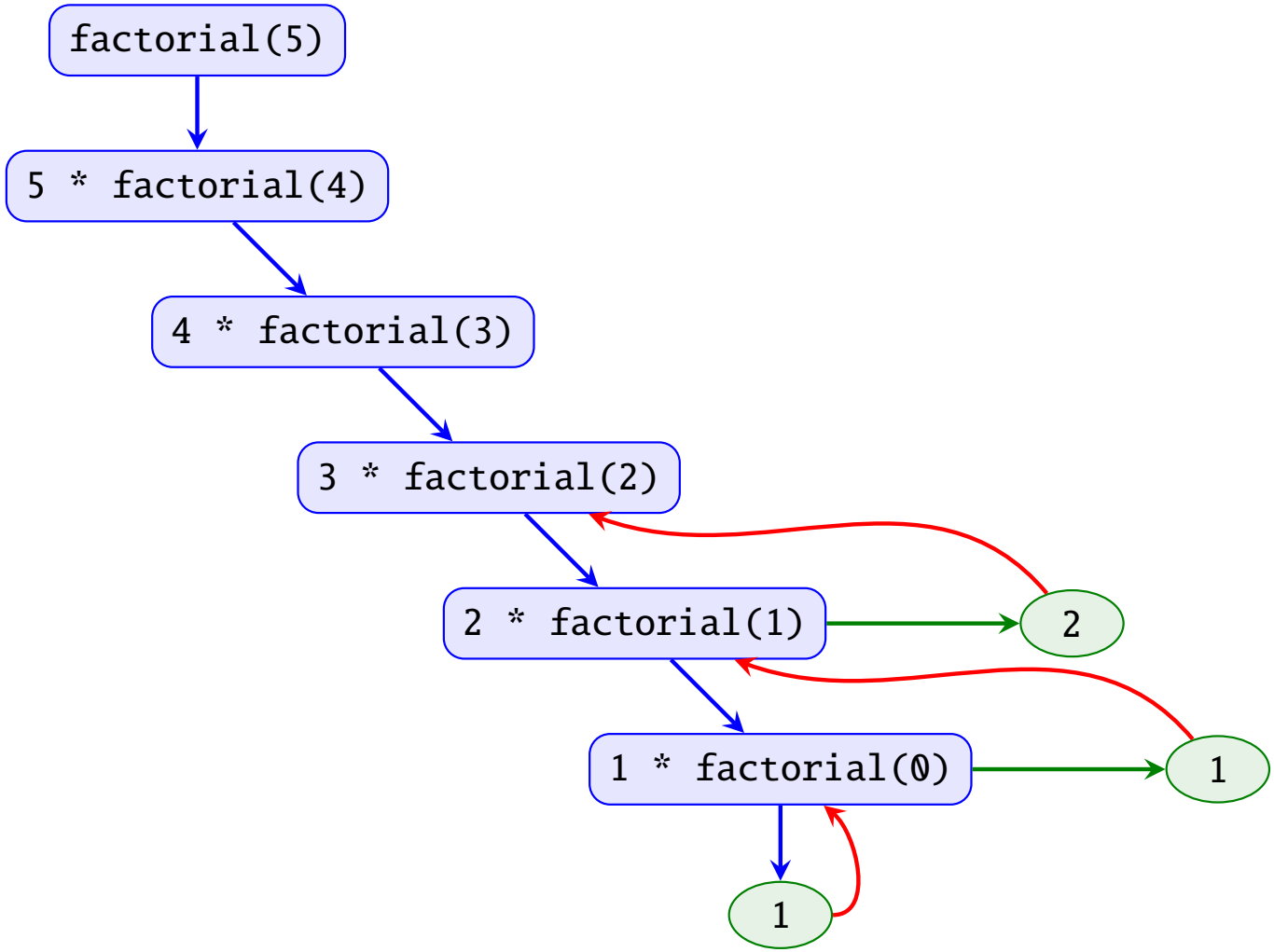
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1 * factorial(0)
1
1

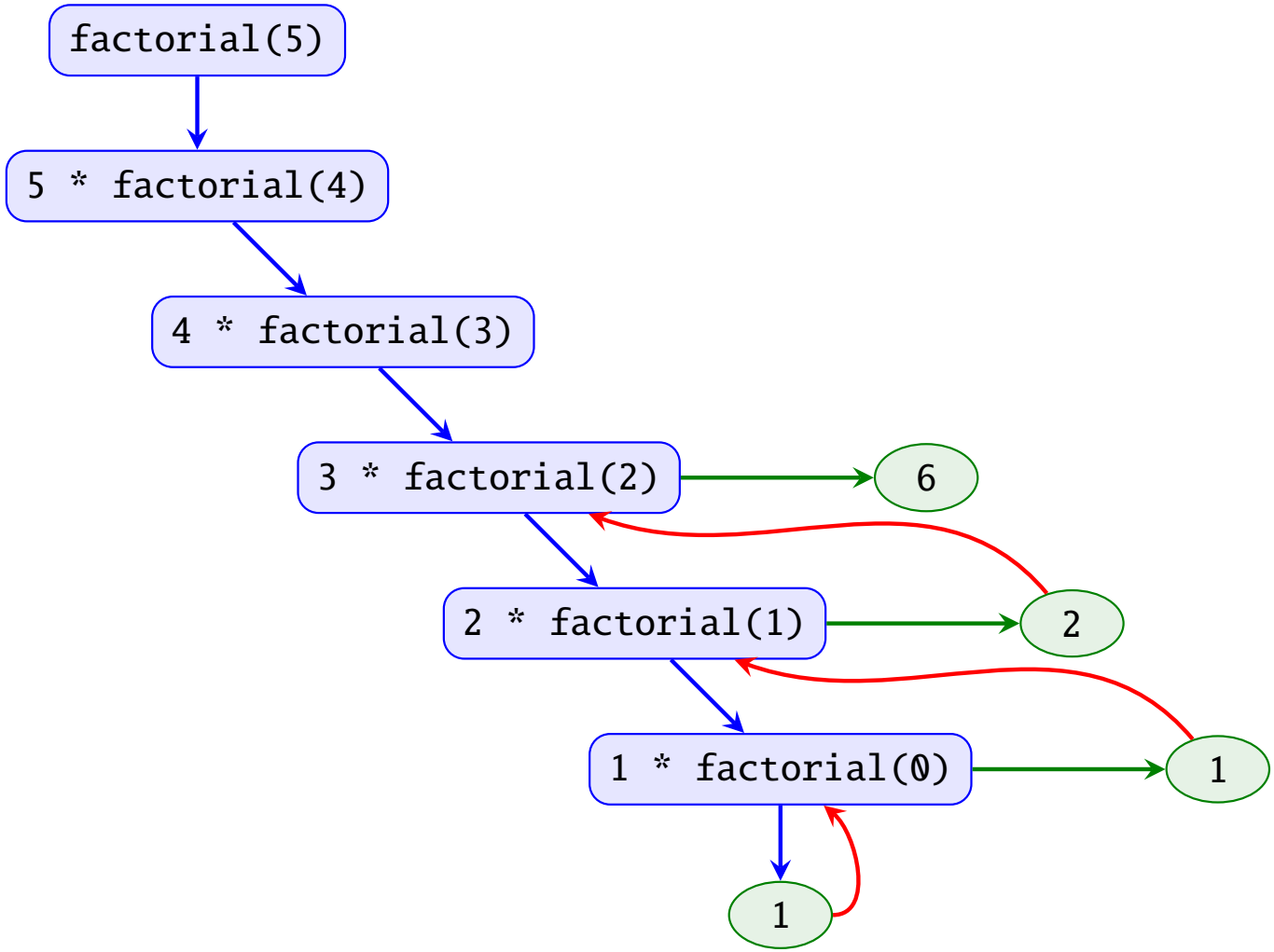
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
2
1 * factorial(0)
1
1

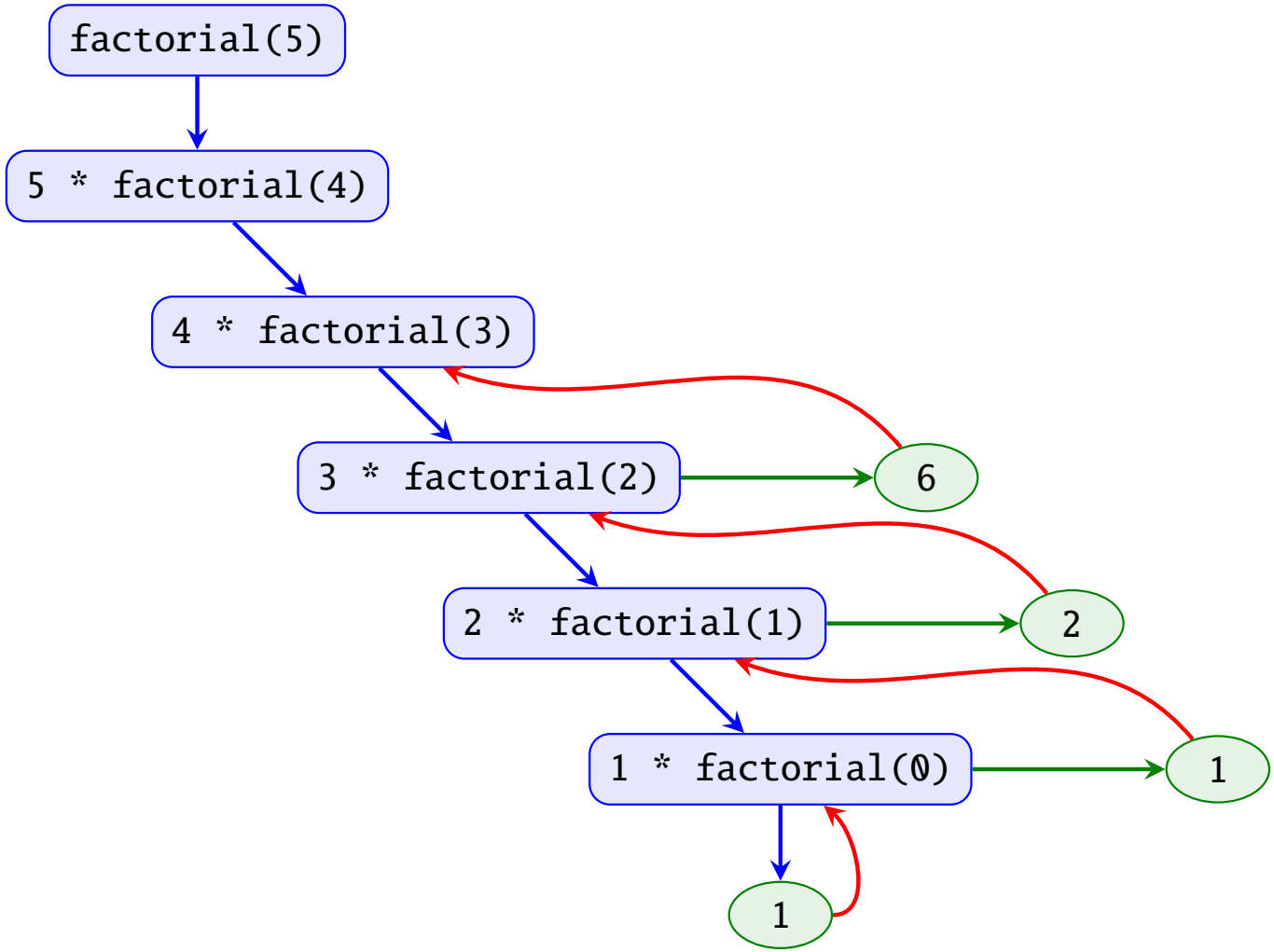
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
2
1 * factorial(0)
1
1

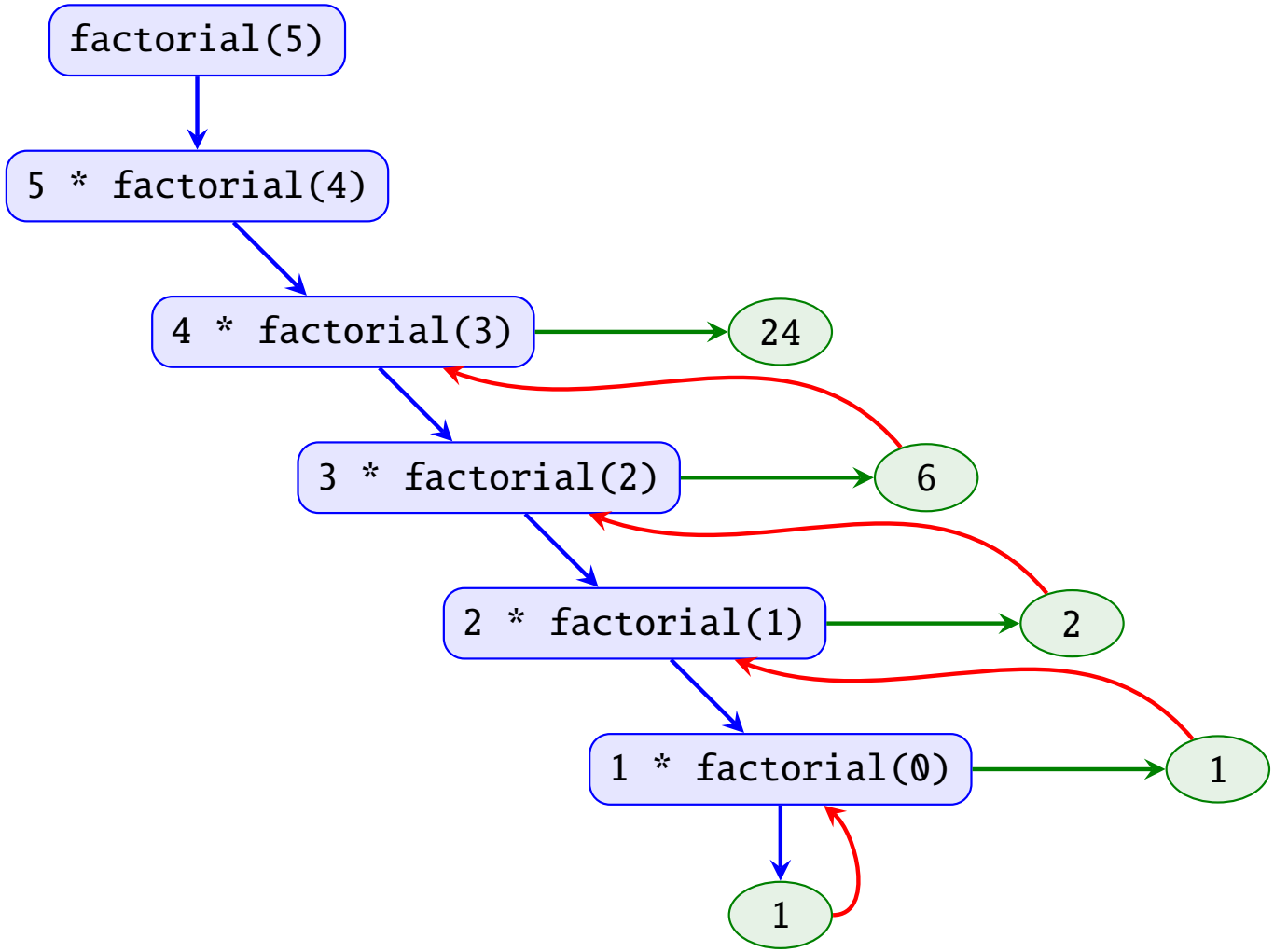
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
6
2 * factorial(1)
2
1 * factorial(0)
1
1

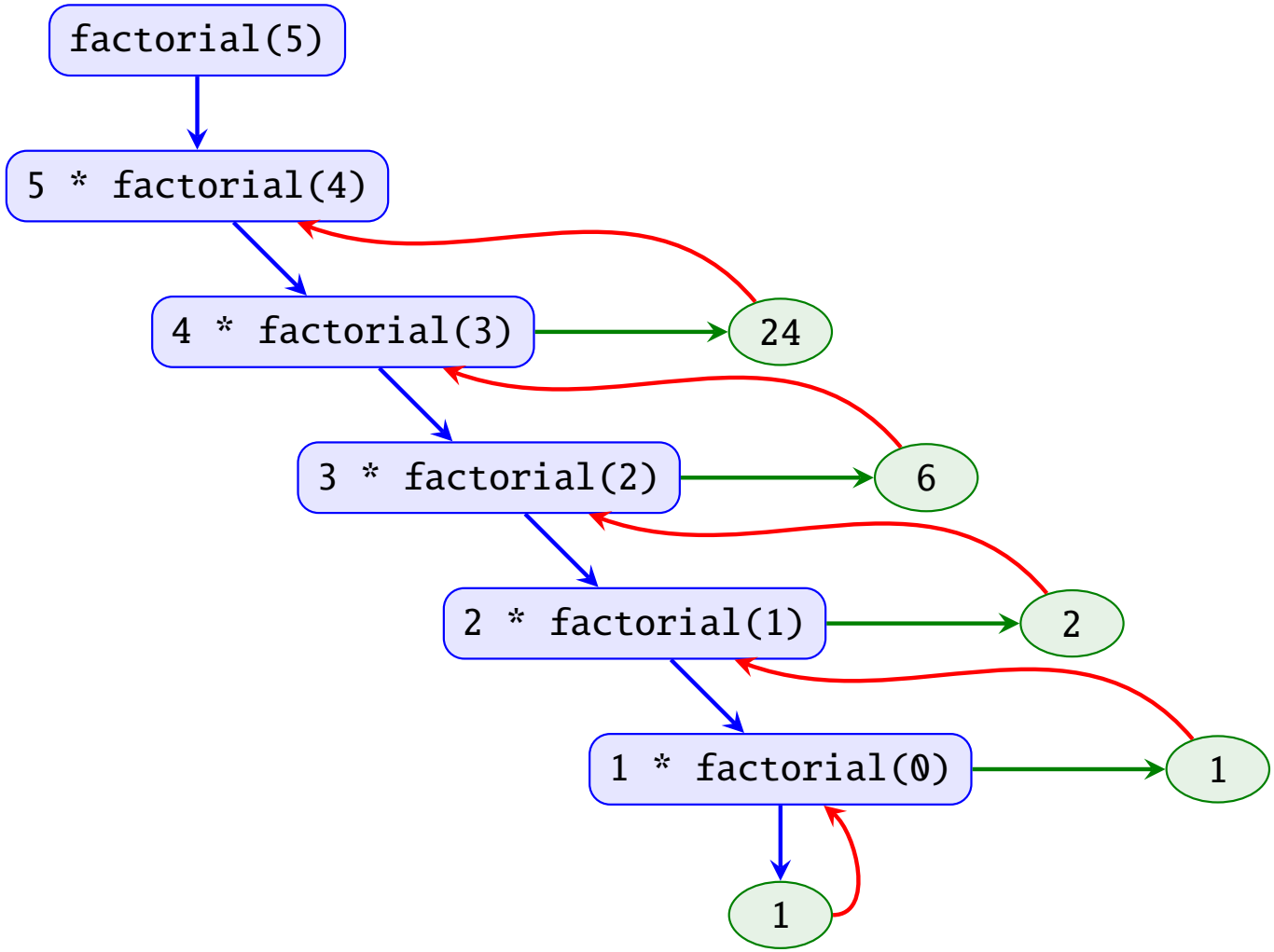
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
6
2 * factorial(1)
2
1 * factorial(0)
1
1

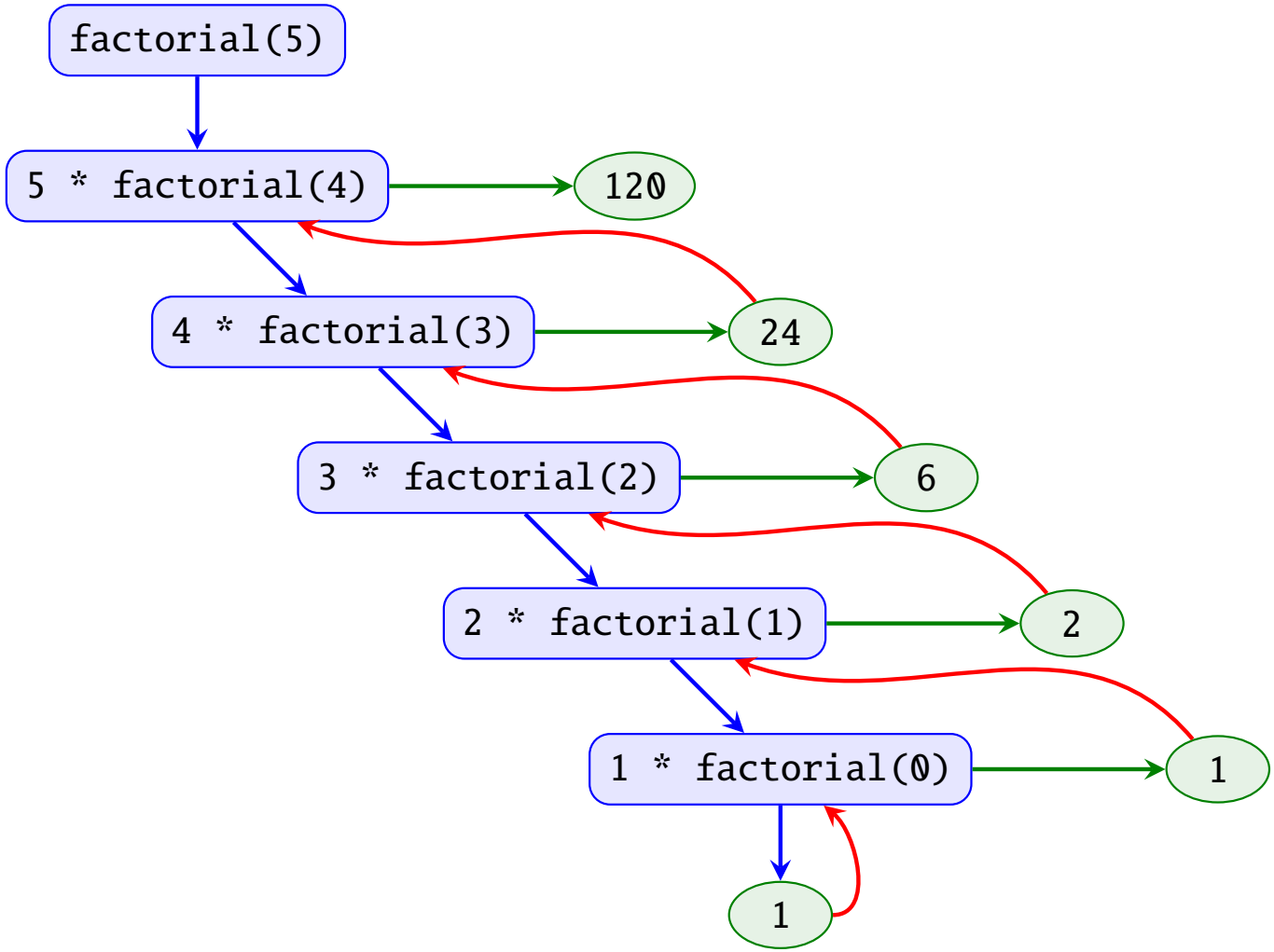
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
24
3 * factorial(2)
6
2 * factorial(1)
2
1 * factorial(0)
1
1

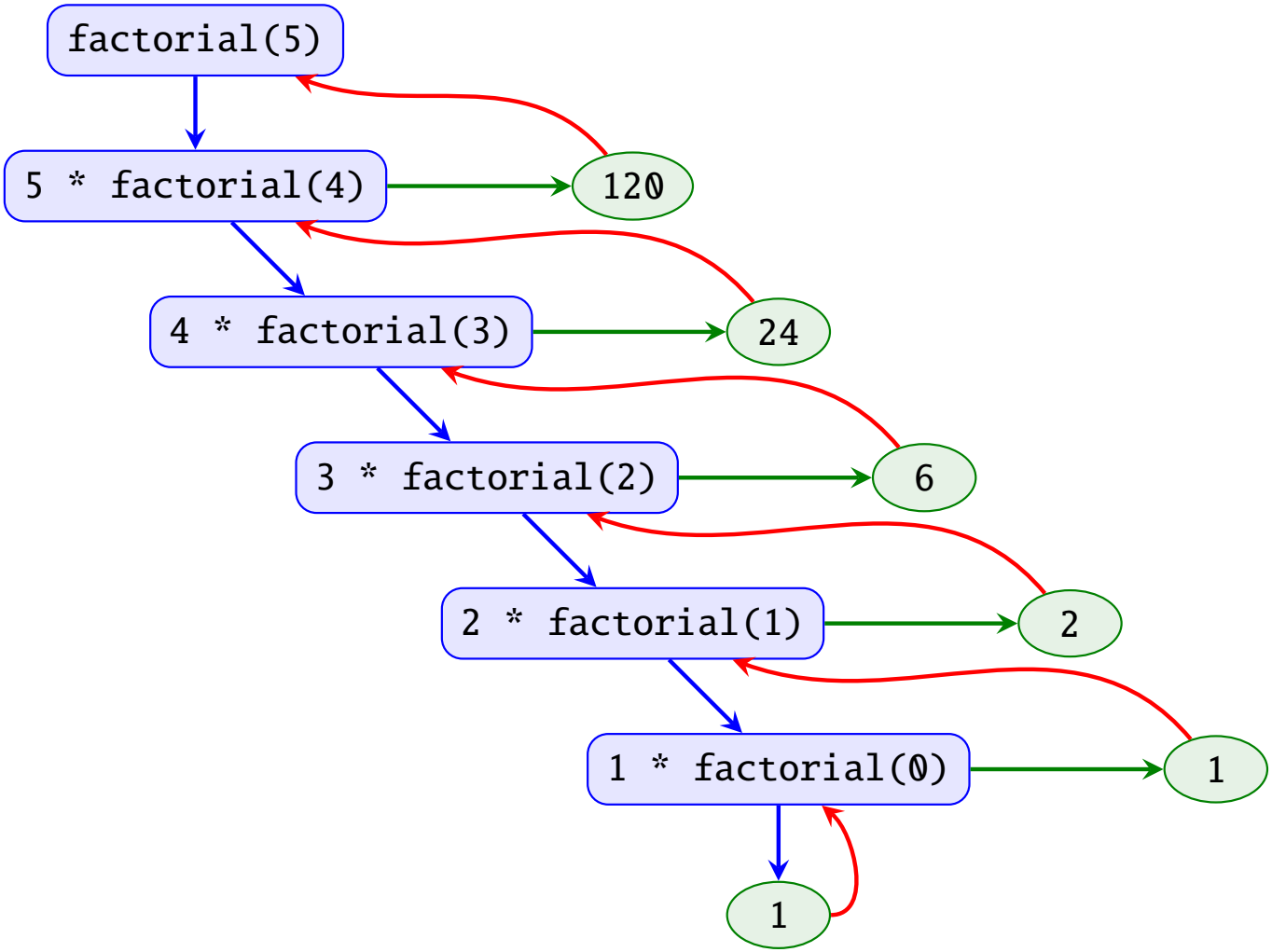
Faktoriaal


factorial(5)
5 * factorial(4)
4 * factorial(3)
24
3 * factorial(2)
6
2 * factorial(1)
2
1 * factorial(0)
1
1

Faktoriaal


factorial(5)
5 * factorial(4)
120
4 * factorial(3)
24
3 * factorial(2)
6
2 * factorial(1)
2
1 * factorial(0)
1
1

Faktoriaal


factorial(5)
5 * factorial(4)
120
4 * factorial(3)
24
3 * factorial(2)
6
2 * factorial(1)
2
1 * factorial(0)
1
1

Rekursioon

Astendamine iteratiivselt

```
def power(n, m):
    p = 1
    for i in range(m):
        p *= n
    return p
```

Astendamine rekursiivselt

```
def power(n, m):
    if m == 0:
        return 1
    else:
        return n * power(n, m-1)
```

Rekursioon

Astendamine iteratiivselt

```
def power(n, m):
    p = 1
    for i in range(m):
        p *= n
    return p
```

Astendamine rekursiivselt

```
def power(n, m):
    if m == 0:
        return 1
    else:
        return n * power(n, m-1)
```

Rekursioon

Listi pikkus iteratiivselt

```
def pikkus(xLst):
    i = 0
    for c in xLst:
        i += 1
    return i
```

Listi pikkus rekursiivselt

```
def pikkus(xLst):
    if xLst == []:
        return 0
    else:
        return 1 + pikkus(xLst[1:])
```

Rekursioon

Listi pikkus iteratiivselt

```
def pikkus(xLst):
    i = 0
    for c in xLst:
        i += 1
    return i
```

Listi pikkus rekursiivselt

```
def pikkus(xLst):
    if xLst == []:
        return 0
    else:
        return 1 + pikkus(xLst[1:])
```

Rekursioon

Palindroomid

```
def palindrome(s):
    if len(s) <= 1:
        return True
    else:
        return s[0] == s[-1] and palindrome(s[1:-1])
```

Näide

```
s = 'kuulilennuteetunneliluuk'
print(palindrome(s))                # Trükib: True
```

Rekursiooni baas ja samm

- Et rekursioon termineeruks, peab rekursiivse funktsiooni kehas toimuma hargnemine sõltuvalt funktsiooni argumentidest.
- Vähemalt üks haru peab olema ilma rekursiivse väljakutseta (nn. rekursiooni baas).
- Rekursiivsed väljakutsed peavad olema "väiksematel" argumentidel.

Näiteid mittetermineeruvast rekursioonist

```
def factorial(n):
    return n * factorial(n-1)      # Rekursiooni baas puudu!
```

Rekursiooni baas ja samm

- Et rekursioon termineeruks, peab rekursiivse funktsiooni kehas toimuma hargnemine sõltuvalt funktsiooni argumentidest.
- Vähemalt üks haru peab olema ilma rekursiivse väljakutseta (nn. rekursiooni baas).
- Rekursiivsed väljakutsed peavad olema "väiksematel" argumentidel.

Näiteid mittetermineeruvast rekursioonist

```
def factorial(n):
    return n * factorial(n-1)      # Rekursiooni baas puudu!
```

Rekursiooni baas ja samm

- Et rekursioon termineeruks, peab rekursiivse funktsiooni kehas toimuma hargnemine sõltuvalt funktsiooni argumentidest.
- Vähemalt üks haru peab olema ilma rekursiivse väljakutseta (nn. rekursiooni baas).
- Rekursiivsed väljakutsed peavad olema "väiksematel" argumentidel.

Näiteid mittetermineeruvast rekursioonist

```
def countUp(n):
    if n <= 0:
        return n
    else:
        return countUp(n+1)      # Argument kasvab!
```

Rekursiooni baas ja samm

Näiteid termineeruvast rekursioonist

```
def foo(n):
    if n >= 20:
        return -1
    else:
        return 2*n + foo(n+4)
```

Rekursiooni baas ja samm

Näiteid termineeruvast rekursioonist

```
def bar(n):
    if n == 0 or n > 20:
        return 1
    else:
        return n * bar(-2*n)
```

Rekursiooni baas ja samm

Kahega jagamine

```
def div2(n):
    if n <= 1:
        return 0
    else:
        return 1 + div2(n-2)
```

Rekursiooni baas ja samm

Suurim ühistegur

```
def gcd(n,m):
    if m == 0:
        return n
    else:
        return gcd(m, n%m)
```

Rekursioon

- Rekursiivsetes funktsioonides võivad "tegevused" toimuda enne ja/või pärast rekursiivset väljakutset.
- Enne rekursiivset väljakutset tehtavad tegevused toimuvad "kahanevalt".
- Pärast rekursiivset väljakutset tehtavad tegevused toimuvad "kasvavalt".

Arvude trükkimine kahanevalt

```
def printDn(n):
    if n <= 0:                  # printDn(4) trükib: 4
        print("Stop!")          #                    3
    else:                       #                    2
        print(n)                #                    1
        printDn(n-1)            #                    Stop!
```

Rekursioon

- Rekursiivsetes funktsioonides võivad "tegevused" toimuda enne ja/või pärast rekursiivset väljakutset.
- Enne rekursiivset väljakutset tehtavad tegevused toimuvad "kahanevalt".
- Pärast rekursiivset väljakutset tehtavad tegevused toimuvad "kasvavalt".

Arvude trükkimine kahanevalt

```
def printDn(n):
    if n <= 0:              # printDn(4) trükib: 4
        print("Stop!")      #                    3
    else:                   #                    2
        print(n)            #                    1
        printDn(n-1)        #                    Stop!
```

Rekursioon

- Rekursiivsetes funktsioonides võivad "tegevused" toimuda enne ja/või pärast rekursiivset väljakutset.
- Enne rekursiivset väljakutset tehtavad tegevused toimuvad "kahanevalt".
- Pärast rekursiivset väljakutset tehtavad tegevused toimuvad "kasvavalt".

Arvude trükkimine kasvavalt

```
def printUp(n):
    if n <= 0:              # printUp(4) trükib: Stop!
        print("Stop!")      #                    1
    else:                   #                    2
        printUp(n-1)        #                    3
        print(n)            #                    4
```

Rekursioon

- Rekursiivsetes funktsioonides võivad "tegevused" toimuda enne ja/või pärast rekursiivset väljakutset.
- Enne rekursiivset väljakutset tehtavad tegevused toimuvad "kahanevalt".
- Pärast rekursiivset väljakutset tehtavad tegevused toimuvad "kasvavalt".

Arvude trükkimine kahanevalt ja seejärel kasvavalt

```
def printDnUp(n):                # printDnUp(3) trükib: 3
    if n <= 0:                   #                      2
        print("Stop!")           #                      1
    else:                        #                      Stop!
        print(n)                 #                      1
        printDnUp(n-1)           #                      2
        print(n)                 #                      3
```

Suur tänu osalemast

ja

kohtumiseni!