Rekurze
(7. přednáška)

FF UK
Department of Logic

Jonathan L. Verner

Rekurze

Rekurzivní definice

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

??? Definice kruhem ???

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

??? Definice kruhem ???
NE!

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

??? Definice kruhem ???
NE!

$5! = 5 \cdot 4!$

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

??? Definice kruhem ???
NE!

$5! = 5 \cdot 4!$, $4! = 4 \cdot 3!$

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

??? Definice kruhem ???
NE!

$5! = 5 \cdot 4!$, $4! = 4 \cdot 3!$, $3! = 3 \cdot 2!$

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

??? Definice kruhem ???
NE!

$5! = 5 \cdot 4!$, $4! = 4 \cdot 3!$, $3! = 3 \cdot 2!$, $2! = 2 \cdot 1!$

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

??? Definice kruhem ???
NE!

$5! = 5 \cdot 4!,\ 4! = 4 \cdot 3!,\ 3! = 3 \cdot 2!,\ 2! = 2 \cdot 1!,\ 1! = 1 \cdot 0!$

FF UK
Department of Logic

$$n! = \begin{cases} 1 & \text{pokud } n < 1 \\ n \cdot (n-1)! & \text{jinak} \end{cases}$$

??? Definice kruhem ???
NE!

$5! = 5 \cdot 4!$, $4! = 4 \cdot 3!$, $3! = 3 \cdot 2!$, $2! = 2 \cdot 1!$, $1! = 1 \cdot 0!$, $\underline{0! = 1}$

Rekurzivní definice

redukce na základní případ

FF UK
Department of Logic

- Rekurzivní definice je zdánlivě kruhová (faktoriál n se definuje pomocí faktoriálu $n - 1$).

Rekurzivní definice

redukce na základní případ

FF UK
Department of Logic

- Rekurzivní definice je zdánlivě kruhová (faktoriál n se definuje pomocí faktoriálu $n - 1$).
- Nicméně dává smysl, neboť:

Rekurzivní definice

redukce na základní případ

FF UK
Department of Logic

- Rekurzivní definice je zdánlivě kruhová (faktoriál n se definuje pomocí faktoriálu $n - 1$).
- Nicméně dává smysl, neboť:
  - Základní případ ($n = 0$) je definován nerekurzivně.

Rekurzivní definice

redukce na základní případ

FF UK
Department of Logic

- Rekurzivní definice je zdánlivě kruhová (faktoriál n se definuje pomocí faktoriálu $n - 1$).
- Nicméně dává smysl, neboť:
  - Základní případ ($n = 0$) je definován nerekurzivně.
  - Všechny ostatní případy se po konečně mnoha krocích převedou na základní případ

Rekurzivní faktoriál v Pythonu

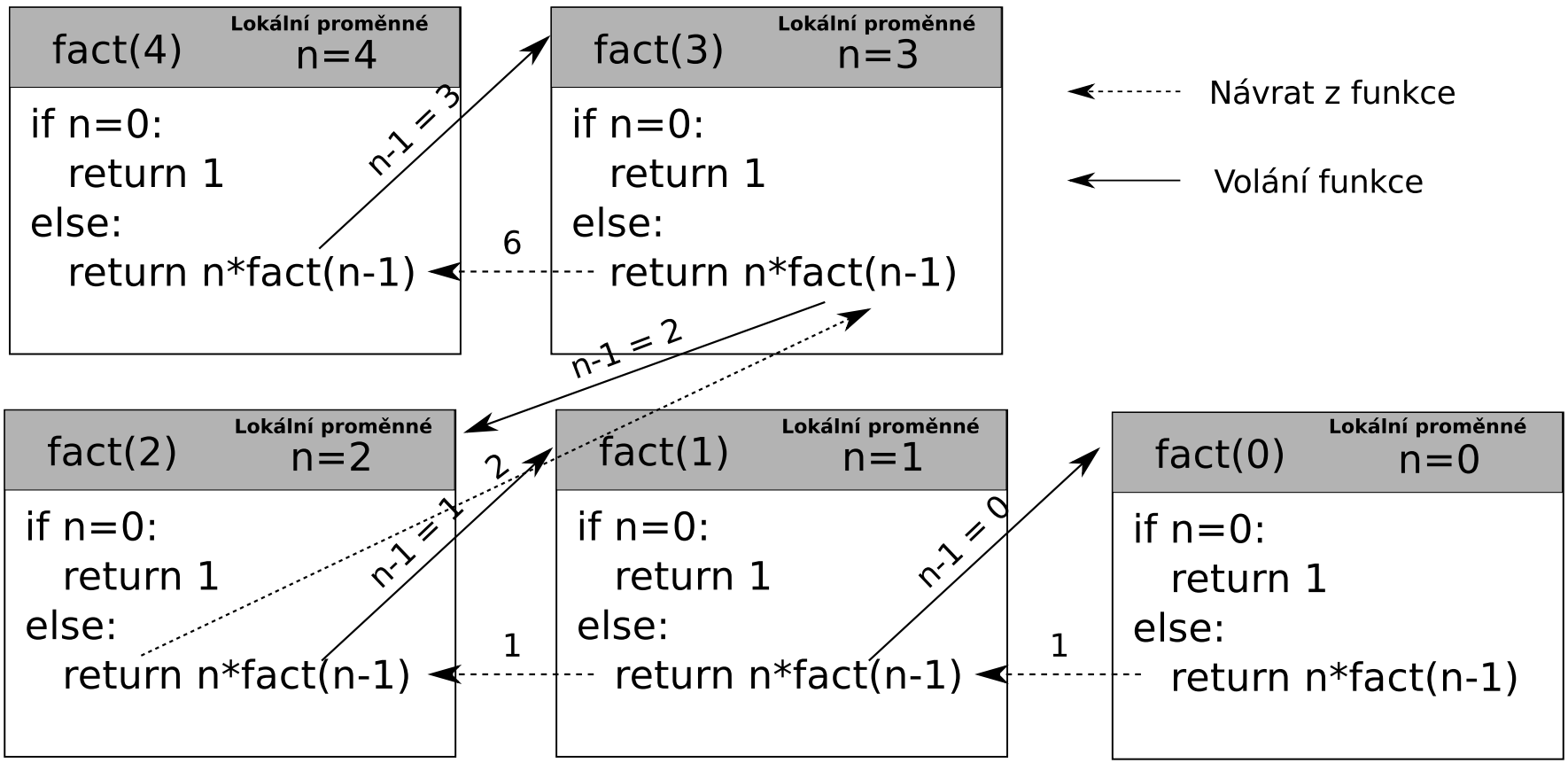
FF UK
Department of Logic

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

```
>>> fact(10)
3628800
```

Rekurzivní faktoriál

volání



Řetězec pozpátku

FF UK
Department of Logic

Napište funkci, která dostane jako parametr řetězec a vrátí ho obrácený.

Řetězec pozpátku

FF UK
Department of Logic

Napište funkci, která dostane jako parametr řetězec a vrátí ho obrácený.

Řešení pomocí `for`-cyklu

Řetězec pozpátku

FF UK
Department of Logic

Napište funkci, která dostane jako parametr řetězec a vrátí ho obrácený.

Řešení pomocí `for`-cyklu

```
def reverse(str):
  ret = ''
  for i in range(len(str)):
    ret += str[-(i+1)]
  return ret
```

Rekurzivní řešení (idea)

Řetězec pozpátku

FF UK
Department of Logic

Napište funkci, která dostane jako parametr řetězec a vrátí ho obrácený.

Řešení pomocí `for`-cyklu

```
def reverse(str):
  ret = ''
  for i in range(len(str)):
    ret += str[-(i+1)]
  return ret
```

Rekurzivní řešení (idea)

$$\mathrm{rev}(c_0 c_1 \cdots c_n) = \begin{cases} c_0 & \text{pokud } n = 0 \\ \mathrm{rev}(c_1 \cdots c_n) c_0 & \text{jinak} \end{cases}$$

Řetězec pozpátku

rekurzivní řešení

FF UK
Department of Logic

```
def reverse(s):
    if len(s) < 2:
        return s
    else:
        return reverse(s[1:])+s[0]
```

```
>>> reverse('ahoj')
'joha'
```

Permutace

FF UK
Department of Logic

Napište funkci, která dostane jako parametr řetězec a vrátí seznam všech možných permutací daného řetězce. Např. pro řetězec `'abc'` vrátí
`['abc','acb','bac','bca','cab','cba']`.

Permutace

FF UK
Department of Logic

Napište funkci, která dostane jako parametr řetězec a vrátí seznam všech možných permutací daného řetězce. Např. pro řetězec `'abc'` vrátí
`['abc','acb','bac','bca','cab','cba']`.

Řešení pomocí knihovny

Permutace

Napište funkci, která dostane jako parametr řetězec a vrátí seznam všech možných permutací daného řetězce. Např. pro řetězec `'abc'` vrátí `['abc','acb','bac','bca','cab','cba']`.

Řešení pomocí knihovny

```
from itertools import permutations

def perm(s):
  ret = []
  for p in permutations(s):
    ret += [''.join(p)]
  return ret
```

Rekurzivní řešení (idea)

FF UK
Department of Logic

Rekurzivní řešení (idea)

- permutace řetězce délky jedna je samotný řetězec

FF UK
Department of Logic

Rekurzivní řešení (idea)

- permutace řetězce délky jedna je samotný řetězec
- je-li řetězec delší, rozdělím si ho na první písmeno a zbytek

FF UK
Department of Logic

Rekurzivní řešení (idea)

- permutace řetězce délky jedna je samotný řetězec
- je-li řetězec delší, rozdělím si ho na první písmeno a zbytek
- vygeneruji si všechny permutace zbytku

Rekurzivní řešení (idea)

- permutace řetězce délky jedna je samotný řetězec
- je-li řetězec delší, rozdělím si ho na první písmeno a zbytek
- vygeneruji si všechny permutace zbytku
- a do každé takto vygenerované permutace budu postupně na všechny možné pozice vkládat první písmeno

Rekurzivní řešení (idea)

- permutace řetězce délky jedna je samotný řetězec
- je-li řetězec delší, rozdělím si ho na první písmeno a zbytek
- vygeneruji si všechny permutace zbytku
- a do každé takto vygenerované permutace budu postupně na všechny možné pozice vkládat první písmeno

Permutace

rekurzivní řešení (implementace)

FF UK
Department of Logic

```
def perm(s):
    if len(s) == 1:
        return [ s ]
    head = s[0]
    tail = s[1:]
    ret = []
    # Projdeme vsechny permutace ocasu
    for p in perm(tail):
        # Na kazdou pozici ve vygenerovane
        # permutaci vlozime head
        for i in range(len(p)+1):
            new_p = p[0:i] + head + p[i:]
            ret += [ new_p ]
    return ret
```

```
>>> perm('abc')
['abc', 'bac', 'bca', 'acb', 'cab', 'cba']
```