

Rekurze  
(7. přednáška)



**FF UK**

Department of Logic

Jonathan L. Verner

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

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

??? Definice kruhem ???

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

??? Definice kruhem ???  
NE!

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

??? Definice kruhem ???  
NE!

$$5! = 5 \cdot 4!$$

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

??? Definice kruhem ???  
NE!

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

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

??? Definice kruhem ???  
NE!

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

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

??? Definice kruhem ???  
NE!

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



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

??? Definice kruhem ???  
NE!

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

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

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

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

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

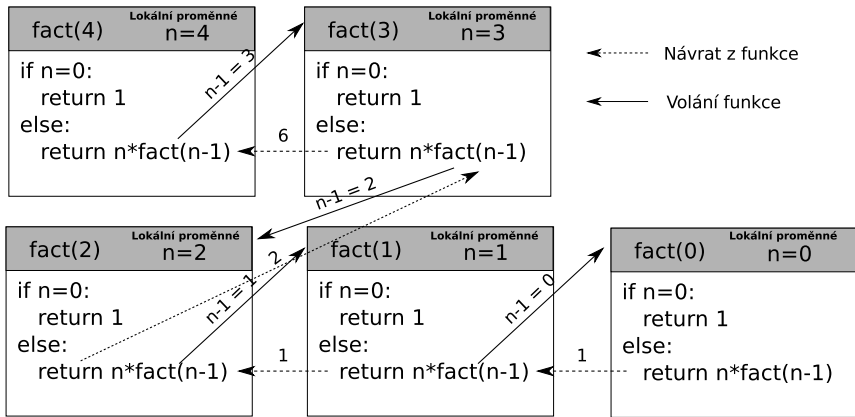
- ▶ Rekurzivní definice je zdánlivě kruhová (faktoriál  $n$  se definuje pomocí faktoriálu  $n - 1$ ).
- ▶ Nicméně dává smysl, neboť:
  - ▶ Základní případ ( $n = 0$ ) je definován nerekurzivně.
  - ▶ Všechny ostatní případy se po konečně mnoha krocích převedou na základní případ

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

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

# Rekurzivní faktoriál

volání





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

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

Řešení pomocí `for`-cyklu

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)

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)

$$\text{rev}(c_0c_1 \dots c_n) = \begin{cases} c_0 & \text{pokud } n = 0 \\ \text{rev}(c_1 \dots c_n)c_0 & \text{jinak} \end{cases}$$

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

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'].
```

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

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)

## Rekurzivní řešení (idea)

- ▶ permutace řetězce délky jedna je samotný řetězec

## Rekurzivní řešení (idea)

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

## 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

```
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']
```