

Rekurzia

Poznámky k téme

Rekurzia znamená, že funkcia volá **samu seba**. Znie to zvláštne, ale veľa problémov sa dá prirodzene opísať práve takto: veľký problém vyriešime pomocou toho istého problému, len menšieho. Faktoriál 5 je 5 krát faktoriál 4; súčet cifier čísla 4728 je 8 plus súčet cifier čísla 472. Kľúčom je vždy vedieť, kedy sa zastaviť.

Čo sa naučíš

- napísať rekurzívnu funkciu so základným prípadom a rekurzívnym krokom,
- pochopiť, prečo rekurzia bez základného prípadu skončí chybou `RecursionError`,
- sledovať priebeh rekurzie (zásobník volaní, poradie výpisov),
- riešiť rekurziou klasické úlohy: súčet cifier, mocninu, NSD, Hanojské veže.

1 Rekurzívna funkcia

Faktoriál sa dá definovať aj takto: $0! = 1$ a $n! = n \cdot (n - 1)!$ pre $n > 0$. Túto definíciu môžeme priamo prepísať do Pythonu:

```
def faktorial(n):  
    if n == 0:                # základný prípad  
        return 1  
    return n * faktorial(n - 1) # rekurzívny krok  
  
print(faktorial(5))  
print([faktorial(i) for i in range(8)])
```

Výstup:

```
120  
[1, 1, 2, 6, 24, 120, 720, 5040]
```

Každá rekurzívna funkcia má dve časti:

1. **Základný prípad** – najjednoduchší vstup, pre ktorý výsledok poznáme hneď, bez ďalšieho volania (`n == 0`).
2. **Rekurzívny krok** – výsledok vyjadríme pomocou volania tej istej funkcie pre **menší** problém (`n - 1`).

Rekurzívny krok musí problém zakaždým **zmenšiť** tak, aby sa raz dostal k základnému prípadu.

2 Základný prípad

Čo sa stane, keď základný prípad chýba?

```
def odpocitaj(n):  
    print(n)  
    odpocitaj(n - 1)  
  
odpocitaj(3)
```

Výstup (program skončí chybou):

```
3  
2  
1  
0  
...  
Traceback (most recent call last):  
  File "program.py", line 5, in <module>  
    odpocitaj(3)  
  File "program.py", line 3, in odpocitaj  
    odpocitaj(n - 1)  
  File "program.py", line 3, in odpocitaj  
    odpocitaj(n - 1)  
  File "program.py", line 3, in odpocitaj  
    odpocitaj(n - 1)  
  [Previous line repeated 992 more times]  
  File "program.py", line 2, in odpocitaj  
    print(n)  
RecursionError: maximum recursion depth exceeded while calling a Python object
```

Funkcia by sa volala donekonečna: 3, 2, 1, 0, -1, -2... Python preto počet vnorených volaní obmedzuje (predvolene asi na 1000) a po jeho prekročení vyvolá `RecursionError`. (Z dlhého výpisu sú tu ukázané iba začiatok a koniec.) Oprava – pridáme základný prípad:

```
def odpocitaj(n):  
    if n == 0:  
        print("Štart!")  
        return  
    print(n)  
    odpocitaj(n - 1)  
  
odpocitaj(3)
```

Výstup:

```
3  
2  
1  
Štart!
```

Rovnako zle dopadne aj funkcia, ktorej krok problém nezmenšuje (`faktorial(n)` volá `faktorial(n)`) alebo ho zmenšuje tak, že základný prípad „preskočí“ (`faktorial(-1)` pri podmienke `n == 0`).

3 Pribeh rekúzie

Pri každom volaní funkcie si Python založí nový „záznam“ s jej lokálnymi premennými. Záznamy sa ukladajú na seba ako taniere – hovoríme o **zásobníku volaní**. Volanie `faktorial(3)` prebieha takto:

```
faktorial(3)
| potrebuje 3 * faktorial(2)
| | potrebuje 2 * faktorial(1)
| | | potrebuje 1 * faktorial(0)
| | | | faktorial(0) vráti 1          ← základný prípad
| | | vráti 1 * 1 = 1
| | vráti 2 * 1 = 2
| vráti 3 * 2 = 6
výsledok 6
```

Najprv ide rekúzia „dole“ až k základnému prípadu, potom sa výsledky vracajú „hore“ a postupne sa skladajú.

3.1 Výpis pred volaním a po ňom

Príkaz **pred** rekúziívnym volaním sa vykoná cestou dole, príkaz **za** ním až cestou späť – v opačnom poradí:

```
def dole_hore(n):
    if n == 0:
        return
    print("dole", n)
    dole_hore(n - 1)
    print("hore", n)

dole_hore(3)
```

Výstup:

```
dole 3
dole 2
dole 1
hore 1
hore 2
hore 3
```

Takto sa dá napríklad vypísať číslo odzadu alebo text obrátene bez cyklu:

```
def odzadu(s):
    if s == "":
        return ""
    return odzadu(s[1:]) + s[0]

print(odzadu("rekúzia"))
```

Výstup:

```
aizruker
```

3.2 Sledovanie hĺbky

Keď chceš rekurzii porozumieť, pridaj do funkcie parameter s hĺbkou a odsadzuj výpis:

```
def sucet(n, hlbka=0):  
    print(" " * hlbka + f"sucet({n})")  
    if n == 0:  
        return 0  
    vysledok = n + sucet(n - 1, hlbka + 1)  
    print(" " * hlbka + f"→ {vysledok}")  
    return vysledok  
  
sucet(3)
```

Výstup:

```
sucet(3)  
  sucet(2)  
    sucet(1)  
      sucet(0)  
    → 1  
  → 3  
→ 6
```

4 Rekurzívne algoritmy

4.1 Súčet cifier

Súčet cifier čísla je jeho posledná cifra plus súčet cifier čísla bez poslednej cifry. Jednociferné číslo je samo sebe súčtom:

```
def sucet_cifier(n):  
    if n < 10:  
        return n  
    return n % 10 + sucet_cifier(n // 10)  
  
print(sucet_cifier(4728))
```

Výstup:

```
21
```

4.2 Mocnina

$a^n = a \cdot a^{n-1}$ a $a^0 = 1$. Rýchlejšia verzia využíva, že pri párnom n je $a^n = (a^{(n/2)})^2$; počet volaní sa tak zmenší z n na približne $\log_2 n$:

```
def mocnina(a, n):
    if n == 0:
        return 1
    return a * mocnina(a, n - 1)

def rychla_mocnina(a, n):
    if n == 0:
        return 1
    polovica = rychla_mocnina(a, n // 2)
    if n % 2 == 0:
        return polovica * polovica
    return a * polovica * polovica

print(mocnina(2, 10), rychla_mocnina(2, 10), rychla_mocnina(3, 13))
```

Výstup:

```
1024 1024 1594323
```

4.3 Najväčší spoločný deliteľ

Euklidov algoritmus je prirodzene rekurzívny: $NSD(a, 0) = a$ a $NSD(a, b) = NSD(b, a \bmod b)$:

```
def nsd(a, b):
    if b == 0:
        return a
    return nsd(b, a % b)

print(nsd(84, 126))
```

Výstup:

```
42
```

4.4 Palindróm

Text je palindróm, ak sa jeho prvý a posledný znak zhodujú a palindrómom je aj to, čo je medzi nimi.

Prázdny text a text s jedným znakom sú palindrómy:

```
def je_palindrom(s):
    if len(s) <= 1:
        return True
    if s[0] != s[-1]:
        return False
    return je_palindrom(s[1:-1])

for slovo in ["kajak", "radar", "python", "a"]:
    print(slovo, je_palindrom(slovo))
```

Výstup:

```
kajak True
radar True
python False
a True
```

4.5 Fibonacciho čísla a cena opakovaných výpočtov

$F(1) = F(2) = 1$ a $F(n) = F(n - 1) + F(n - 2)$. Priamy rekurzívny zápis je krásne krátky, ale veľmi pomalý: tie isté hodnoty počíta znova a znova ($F(5)$ volá $F(4)$ aj $F(3)$, $F(4)$ opäť $F(3)$...). Spočítajme volania:

```
volania = 0

def fib(n):
    global volania
    volania += 1
    if n <= 2:
        return 1
    return fib(n - 1) + fib(n - 2)

for n in [10, 20, 25]:
    volania = 0
    print(f"fib({n}) = {fib(n)}, počet volaní: {volania}")
```

Výstup:

```
fib(10) = 55, počet volaní: 109
fib(20) = 6765, počet volaní: 13529
fib(25) = 75025, počet volaní: 150049
```

Počet volaní rastie takmer dvojnásobne s každým ďalším n . Cyklom (téma Cykly) sa $F(n)$ vypočíta v n krokoch. Rekúziu preto používaj tam, kde sa podproblémy neopakujú – alebo si už vypočítané výsledky pamätaj (napríklad v slovníku).

4.6 Hanojské veže

Na prvom kolíku je veža z n diskov (väčšie dole). Treba ju presunúť na tretí kolík; naraz sa smie presúvať iba jeden disk a nikdy nesmie ležať väčší disk na menšom. Riešenie je typicky rekurzívne:

1. presuň $n - 1$ diskov z prvého kolíka na pomocný,
2. presuň najväčší disk na cieľový kolík,
3. presuň $n - 1$ diskov z pomocného kolíka na cieľový.

```
def hanoj(n, odkial, kam, pomocny):
    if n == 0:
        return
    hanoj(n - 1, odkial, pomocny, kam)
    print(f"disk {n}: {odkial} → {kam}")
    hanoj(n - 1, pomocny, kam, odkial)

hanoj(3, "A", "C", "B")
```

Výstup:

```
disk 1: A → C
disk 2: A → B
disk 1: C → B
disk 3: A → C
disk 1: B → A
disk 2: B → C
disk 1: A → C
```

Na n diskov treba $2^n - 1$ presunov – pre 3 disky 7, pre 64 diskov viac, než by stihol ktokoľvek za celý život.

4.7 Všetky možnosti: dvojkové reťazce

Rekúzia sa výborne hodí na **vymenovanie všetkých možností**: na každom mieste skúsime 0 aj 1 a pre zvyšok zavoláme tú istú funkciu.

```
def vsetky(n, zaciatok=""):
    if n == 0:
        print(zaciatok, end=" ")
        return
    vsetky(n - 1, zaciatok + "0")
    vsetky(n - 1, zaciatok + "1")

vsetky(3)
print()
```

Výstup:

```
000 001 010 011 100 101 110 111
```

4.8 Súčet čísel vo vnorenom zozname

Zoznam môže obsahovať ďalšie zoznamy, tie zase ďalšie... Rekúzia si s ľubovoľnou hĺbkou vnorenia poradí ľahko:

```
def sucet(z):
    spolu = 0
    for prvok in z:
        if isinstance(prvok, list):
            spolu += sucet(prvok)      # vnorený zoznam – rekúzia
        else:
            spolu += prvok
    return spolu

print(sucet([1, [2, 3], [4, [5, 6, [7]]], 8]))
```

Výstup:

```
36
```

`isinstance(prvok, list)` zisťuje, či je prvok zoznam.

5 Rekúzia alebo cyklus?

- Každú rekúziu možno prepísať na cyklus a naopak.
- Rekúzia je prirodzená pri problémoch, ktoré sa rozpadajú na menšie problémy rovnakého druhu: vnorené štruktúry, stromy, „rozdeľ a panuj“ (binárne vyhľadávanie, triedenie zlučovaním), vymenovanie všetkých možností.
- Cyklus býva rýchlejší a nemá obmedzenie hĺbky. Na jednoduché opakovanie (súčet 1 až n, faktoriál) je v Pythone lepší cyklus.

6 Časté chyby

- **Chýbajúci alebo nedosiahnuteľný základný prípad** → `RecursionError`.
- **Krok nezmenšuje problém** (volanie s rovnakým argumentom).
- **Zabudnutý return pri rekúziínom volaní**: `faktorial(n - 1)` bez `return` výsledok zahodí a funkcia vráti `None`.

```
def faktorial(n):
    if n == 0:
        return 1
    n * faktorial(n - 1)      # chyba return

print(faktorial(3))
```

Výstup (program skončí chybou):

```
Traceback (most recent call last):
  File "program.py", line 6, in <module>
    print(faktorial(3))
    ~~~~~^
  File "program.py", line 4, in faktorial
    n * faktorial(n - 1)      # chyba return
    ~~~~~^
  File "program.py", line 4, in faktorial
    n * faktorial(n - 1)      # chyba return
    ~~~~~^
TypeError: unsupported operand type(s) for *: 'int' and 'NoneType'
```

Výnimka vznikla až v hĺbke: `faktorial(1)` vrátil `None` a výraz `2 * None` je chyba.

- **Príliš veľký vstup:** `faktorial(5000)` prekročí limit hĺbky – použi cyklus.

Zhrnutie

- Rekurzívna funkcia volá samu seba; má **základný prípad** (bez volania) a **rekurzívny krok** (menší problém).
- Bez dosiahnuteľného základného prípadu vznikne `RecursionError` (limit asi 1000 volaní).
- Volania tvoria zásobník: cestou dole sa volá, cestou hore sa výsledky skladajú; príkazy za volaním sa vykonávajú v opačnom poradí.
- Klasika: faktoriál, súčet čísel, mocnina, NSD, palindróm, Hanojské veže, vymenovanie možností.
- Pri opakujúcich sa podproblémoch (naivný Fibonacci) je rekúzia veľmi pomalá.

Úlohy na precvičenie

1. Napíš rekurzívnu funkciu, ktorá vráti súčet čísel od 1 po `n`.
2. Napíš rekurzívnu funkciu `pocet_cifier(n)`.
3. Napíš rekurzívnu funkciu, ktorá vypíše čísla od `n` po 1 a potom späť od 1 po `n`.
4. Napíš rekurzívnu funkciu, ktorá vráti maximum zoznamu (maximum z prvého prvku a maxima zvyšku).
5. Uprav Fibonacciho funkciu tak, aby si vypočítané hodnoty pamätala v slovníku, a porovnaj počet volaní.
6. Vypíš všetky slová dĺžky 3 z písmen `a`, `b`, `c` (rekurzívne, ako dvojkové reťazce).

Úlohy k tejto téme nájdeš na learnwithmente.com v časti Informatika → Python.