Co je rekurze?

Rekurze je, když funkce **volá sama sebe**, aby vyřešila menší podproblém původního problému. Funguje to podobně jako když se díváš do dvou zrcadel naproti sobě – vidíš nekonečný odraz. Ale v programování musíš **určit konečný bod**, tzv. **podmínku ukončení (base case)**, jinak by se funkce volala donekonečna.

Jednoduchý příklad v Pythonu – faktoriál

Faktoriál čísla `n` (značí se `n!`) je součin všech celých čísel od 1 do `n`.
Např.:
`5! = 5 × 4 × 3 × 2 × 1 = 120`

Rekurzivní verze:

python Copy Edit

```python
def faktorial(n):
    if n == 0 or n == 1:  # podmínka ukončení
        return 1
    else:
        return n * faktorial(n - 1)  # rekurzivní volání

print(faktorial(5))  # výstup: 120
```

Jak to funguje krok za krokem:

- `faktorial(5)` → `5 * faktorial(4)`
- `faktorial(4)` → `4 * faktorial(3)`
- `faktorial(3)` → `3 * faktorial(2)`
- `faktorial(2)` → `2 * faktorial(1)`
- `faktorial(1)` → `1` (tady se to zastaví a začne to „vracet“ hodnoty zpět nahoru)

Můžeš nám vysvětlit prosím rekurzi pro mazání souboru? Proč je potřeba?

Jasně, výborná otázka! Rekurze se při **mazání souborů** používá hlavně tehdy, když **mažeme složku, která obsahuje další podsložky a soubory**. Není to tedy o mazání jednoho souboru, ale **stromové struktury** – a tam rekurze skvěle sedí.

Proč je rekurze při mazání složek potřeba?

Složky (adresáře) se mohou vnořovat – tj. složka může obsahovat jinou složku, která zase obsahuje další...
Abys mohla smazat hlavní složku, musíš **nejdřív smazat vše uvnitř**.
A to je ideální situace pro rekurzi:

- podíváš se, co je uvnitř složky
- pokud je to soubor, smažeš ho
- pokud je to další složka, zavoláš **stejnou funkci znovu** – a ta řeší to samé, jen na vnořenější úrovni

Příklad v Pythonu – rekurzivní mazání složky

python Copy Edit

```python
import os

def smaz_slozku_celkove(cesta):
    if os.path.isdir(cesta):
        for polozka in os.listdir(cesta):
            plna_cesta = os.path.join(cesta, polozka)
            smaz_slozku_celkove(plna_cesta)  # rekurzivní volání pro podsložky a so
        os.rmdir(cesta)  # smaže prázdnou složku
    else:
        os.remove(cesta)  # smaže soubor
```

Použití:

python Copy Edit

```python
smaz_slozku_celkove("/cesta/k/moji/slozce")
```

Jak to funguje?

1. Funkce nejprve zjistí, zda je zadaná cesta složka (`os.path.isdir`).
2. Pokud ano, projde její obsah (`os.listdir`) a na každou položku zavolá **sama sebe** – to je ta rekurze!
3. Tím se dostane až k nejvnitřnějším souborům (které smaže).
4. Poté se vrací zpět nahoru a maže prázdné složky.

Proč nestačí `os.rmdir` nebo `os.remove` ?

- `os.remove()` smaže jen **soubor**
- `os.rmdir()` smaže jen **prázdnou složku**
- ale reálné složky často **obsahují další věci**, takže bez rekurze to nepůjde

Co je to jmenná konvence souborů jako `*.txt` ?

Zápis `*.txt` znamená:

- `*` = **jakýkoli název souboru**
- `.txt` = **koncovka souboru** (přípona), v tomto případě označující **textový soubor**

Takže `*.txt` znamená **všechny soubory, které končí na** `.txt` , například:

- `poznamky.txt`
- `seznam.txt`
- `uloha1.txt`

Používá se to často při hledání nebo filtrování souborů ve složce – např. ve skriptech nebo při mazání, kopírování, načítání souborů určitého typu.

Další běžné přípony a co znamenají:

Přípona	Typ souboru	Vysvětlení
`.txt`	Textový soubor	Obsahuje čistý text, čitelný pro člověka
`.csv`	Comma Separated Values	Textový soubor s daty oddělenými čárkami (nebo středníky)
`.json`	JavaScript Object Notation	Strukturovaný datový formát, čitelný pro stroj i člověka
`.py`	Python script	Zdrojový kód v jazyce Python
`.log`	Logovací soubor	Záznamy událostí nebo výpisy programů
`.jpg`, `.png`, `.gif`	Obrázky	Různé formáty pro grafické soubory
`.pdf`	Portable Document Format	Dokumenty (často finální formát pro sdílení)
`.zip`, `.rar`	Komprimované soubory	Archivované složky/soubory

Jak použít `*.txt` v Pythonu

Například chceš načíst všechny `.txt` soubory ve složce:

python Copy Edit

```python
import glob

txt_soubory = glob.glob("*.txt")  # vypíše seznam všech .txt souborů v aktuální slo
print(txt_soubory)
```

Bonus – co je `*` a jak funguje:

`*` je **wildcard** (zástupný znak) a znamená „cokoli“ (0 a více znaků).
Například:

- `*.py` = všechny Python soubory
- `data*` = všechny soubory začínající na „data“
- `*log*` = všechny soubory, které **obsahují** slovo „log“

Jak funguje `[]` v názvech souborů

Uvnitř `[]` specifikuješ **sadu povolených znaků** pro **jednu pozici** v názvu souboru.

Příklady:

- `file[1-3].txt`
  odpovídá: `file1.txt`, `file2.txt`, `file3.txt`
- `test[ab].py`
  odpovídá: `testa.py`, `testb.py`
- `data[0-9].csv`
  odpovídá souborům jako: `data0.csv`, `data1.csv`, ..., `data9.csv`

V Pythonu s `glob`:

python Copy Edit

```python
import glob

# všechny soubory file1.txt, file2.txt, file3.txt
soubory = glob.glob("file[1-3].txt")
print(soubory)
```

Shrnutí zástupných znaků (wildcards):

Symbol	Význam
`*`	libovolný počet **jakýchkoli znaků** (vč. žádného)
`?`	**právě jeden** libovolný znak
`[abc]`	**jeden** znak z množiny `a`, `b`, nebo `c`
`[a-z]`	**jeden** znak z rozsahu `a` až `z`
`[!x]` nebo `[^x]`	**negace** – jakýkoli znak kromě `x`

Příklad kombinace:

```python
glob.glob("data[0-9]*.csv")
```

Najde třeba:

- `data1.csv`
- `data2_backup.csv`
- `data3_2024.csv`

Co je to encoding (kódování)?

Encoding (kódování) určuje, **jak se znaky ukládají do souboru jako bajty**.
Počítače nerozumí písmenkům nebo znakům – vše musí být převedeno na **čísla / bajty**.

Co je UTF-8?

`UTF-8` je zkratka pro **Unicode Transformation Format – 8bit**.
Je to **nejpoužívanější formát** kódování textu na světě (včetně webu, Pythonu atd.).

- **Podporuje všechny znaky Unicode** (např. české znaky, emoji, čínské znaky, atd.)
- Je **zpětně kompatibilní s ASCII** (tedy běžné znaky jako `A-Z`, `0-9`, `.` atd.)
- Ukládá znaky jako **proměnný počet bajtů** (1–4 bajty podle složitosti znaku)

Příklady:

Znak	Unicode kód	UTF-8 bajty
`A`	U+0041	1 bajt
`č`	U+010D	2 bajty
😊	U+1F60A	4 bajty

Proč na tom záleží?

Když otevřeš nebo uložíš soubor ve **špatném kódování**, text se může „rozbít“ (např. místo `č` se objeví `` nebo nesmyslné znaky).

V Pythonu – použití UTF-8 při čtení/zápisu souborů:

python Copy Edit

```python
# Čtení souboru s UTF-8
with open("soubor.txt", encoding="utf-8") as f:
    obsah = f.read()

# Zápis souboru s UTF-8
with open("vystup.txt", "w", encoding="utf-8") as f:
    f.write("čau světe! 😊")
```

Shrnutí výhod UTF-8:

- **Univerzální** – funguje s jakýmkoli jazykem (česky, japonsky, emoji…)
- **Efektivní** – pro běžný text zabírá málo místa
- **Bezpečný** – zabraňuje chybám s diakritikou
- **Standardní** – výchozí ve většině nástrojů, webu i Pythonu

Má Python výchozí (defaultní) kódování?

Ano, **má**, ale **liší se podle prostředí a verze Pythonu**:

Od Pythonu 3.7 a vyšších:

- Výchozí kódování při práci se soubory je **platform-dependent**:
  - Na **Windows** to bývá často `cp1250` nebo `cp1252` (Windows-1250/1252 – středoevropské kódování)
  - Na **Linuxu/macOS** to bývá `utf-8`

Ale pozor:

- **Vždy je lepší kódování určit ručně**, protože jinak můžeš narazit na problémy s diakritikou, zejména když:
  - kód poběží na jiném OS
  - načítáš soubory od jiných lidí
  - ukládáš české znaky (č, ř, ž…)

Doporučený způsob (vždy zadat `encoding="utf-8"`):

python Copy Edit

```python
with open("soubor.txt", "w", encoding="utf-8") as f:
    f.write("čau světe")
```

Jak zjistit výchozí kódování tvého Pythonu:

python Copy Edit

```python
import locale
print(locale.getpreferredencoding(False))
```

Nebo:

python Copy Edit

```python
import sys
print(sys.getdefaultencoding())
```

Shrnutí:

Otázka	Odpověď
Má Python výchozí kódování?	Ano, ale není vždy `utf-8`
Musím zadávat `encoding="utf-8"`?	**Doporučuje se ano** – pro jistotu a přenositelnost
Kdy to opravdu vadí?	Při práci s diakritikou, mezi systémy, nebo když čteš/zapisuješ cizí soubory

Co ještě znamená ve for když prohledávám a zadám topdown=False?

Skvělý dotaz! `topdown=False` se používá při **procházení adresářů** – typicky s funkcí `os.walk()`.

Co dělá `os.walk()`?

Funkce `os.walk()` prochází celou složkovou strukturu od zadané cesty – buď:

- **shora dolů** (`topdown=True`, což je výchozí)
- nebo **zdola nahoru** (`topdown=False`)

🔁 `os.walk()` – co dělá:

Funkce `os.walk()` prochází **adresářový strom** a při každé iteraci ti vrátí:

- `root`: aktuální složku
- `dirs`: seznam podsložek
- `files`: seznam souborů

🧭 Co znamená `topdown=True` (výchozí chování):

- Prochází **odshora dolů** – tedy nejprve navštíví **nadřazenou složku**, a pak její podsložky.

```python
import os

for root, dirs, files in os.walk("moje_slozka", topdown=True):
    print("Procházím složku:", root)
```

Vhodné, když chceš:

- **měnit seznam** `dirs` za běhu – např. přeskočit určité složky
- nejdřív pracovat se složkami, pak se soubory uvnitř

🔄 `topdown=False`:

- Prochází **odspodu nahoru** – nejprve nejvnitřnější složky, až nakonec nadřazené.

```python
for root, dirs, files in os.walk("moje_slozka", topdown=False):
    print("Procházím složku:", root)
```

Vhodné, když chceš:

- **mazat složky** – protože musíš nejprve smazat obsah a teprve pak samotnou složku (klasický důvod!)
- nebo řešíš něco, co závisí na tom, co je **uvnitř složek**

💡 Mini příklad rozdílu:

Struktura složky:

lua Copy Edit

```
projekt/
├── data/
│   └── soubor.csv
└── log/
    └── app.log
```

`topdown=True`:

Projde složky v tomto pořadí:

1. `projekt/`
2. `projekt/data/`
3. `projekt/log/`

`topdown=False`:

Projde složky v opačném pořadí:

1. `projekt/data/`
2. `projekt/log/`
3. `projekt/`

🧹 Skript na mazání pouze prázdných složek:

python Copy Edit

```python
import os

def smaz_prazdne_slozky(cesta):
    for root, dirs, files in os.walk(cesta, topdown=False):
        # pokud složka nemá žádné soubory ani podsložky
        if not dirs and not files:
            try:
                os.rmdir(root)
                print(f"Smazaná prázdná složka: {root}")
            except OSError as e:
                print(f"❌ Nelze smazat {root}: {e}")

# Zavolej funkci s cestou ke kořenové složce
smaz_prazdne_slozky("moje_slozka")
```

📝 Co skript dělá:

- `os.walk(..., topdown=False)` → projde složky **od nejvnitřnějších**
- Pro každou složku se zkontroluje, zda:
  - neobsahuje **soubory** (`files == []`)
  - neobsahuje **podsložky** (`dirs == []`)
- Pokud je opravdu prázdná → smaže se pomocí `os.rmdir()`

✅ Příklad použití:

Dejme tomu, že máš složku `projekt/` a v ní jsou už **některé prázdné složky** – spustíš tento skript a on je bezpečně odstraní, aniž by sáhl na složky, které obsahují něco důležitého.