

# Legal Tech: I

Jan Černý, Tomáš Knap

**crimczech**

Charles University

**Version 1.0, 2026**

## Anonymní feedback



Naskenujte telefonem nebo **klikněte**: dvě minuty, bez jména

## Jak na úkoly

**Pár rad, než se pustíte do dalších úloh**

# Obecné rady k úkolům

Všechny úlohy jdou vyřešit **jen syntaxí, kterou jsme probrali**; když vám řešení vychází složitě, hledáte nejspíš špatnou cestu

Nedělejte **nic navíc**, než po vás chce zadání: žádné bonusové funkce, menu ani ošetření situací, které zadání nezmiňuje

**Zadání čtěte doslovně**, jako zákon: každé slovo v něm je z nějakého důvodu

Když si nejste jistí, jak zadání chápat, **zeptajte se**; ušetříte si tím čas

Odevzdávejte kód, který jste **spustili** a který na příkladech ze zadání dělá, co má

# Jak postupovat při řešení úloh

1. **Přečíst a pochopit zadání:** co je vstup, co má být výstup, co se má stát v krajních případech
2. **Rozmyslet** postup bez počítače: na papír, slovy nebo v krocích, jako byste ho vysvětlovali kolegovi
3. **Sepsat** kód po malých kusech a každý kus hned spustit
4. **Zamyslet se nad edge casey:** prázdný vstup, nula, záporné číslo, jeden prvek, hodně dlouhý text
5. **Zkontrolovat konvence a okomentovat:** výstižná jména, typové anotace, komentář tam, kde vysvětluje proč

Na pořadí záleží: kdo začne psát kód bez bodů 1 a 2, píše ho obvykle dvakrát

## Rada

Python dovolí spoustu věcí, ale to neznamená, že je dobrý nápad je dělat.

## Metafora

Do topinkovače jde strčit spousta jiných věcí než chleba, ale to neznamená, že je to dobrý nápad.

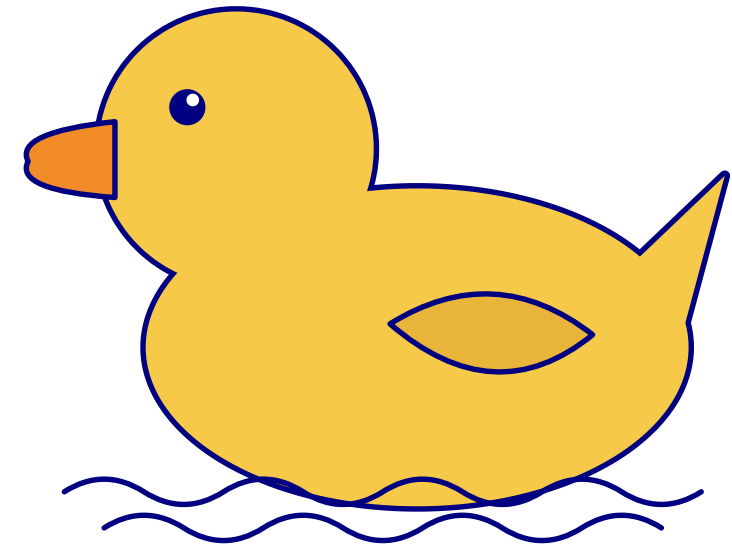
- Kód čteme mnohem častěji, než ho píšeme: za týden vy, při opravě vyučující
- Funkční kód je *minimum*; kód, kterému hned rozumí i někdo jiný, je *cíl*

## Zde jsme sepsali rady, jak psát čitelný kód

- Jména, odsazení, mezery, uvozovky, typové anotace, komentáře: vše na jednom místě
- Krok 5 z postupu řešení úloh: před odevzdáním si podle nich kód projděte
- Oficiální a mnohem delší průvodce stylem Pythonu je [PEP 8](#)
- Naskenujte, nebo klikněte: [ggu.cz/python-conventions](https://ggu.cz/python-conventions)



- Používejte zabudovaný debugger ve **VS Codium**
  - [video návod](#)
- Rubber duck debugging
  - vysvětlíte svůj kód **nahlas, řádek po řádku**  
gumové kachničky (nebo komukoliv, kdo vydrží poslouchat)
  - chybu obvykle najdete sami během vysvětlování:  
nutí vás říct, co řádek *opravdu* dělá, ne co měl dělat



# Datové struktury

Dejme datům řád

# Datová struktura: co to je

## Datová struktura

Způsob, jak v programu **uspořádat více hodnot dohromady** tak, aby se v nich dalo rychle hledat, přidávat do nich a procházet je.

- Zatím jedna proměnná drží **jednu hodnotu**: číslo, text, `bool`. Smluv, lhůt nebo paragrafů je ale mnoho a patří k sobě
- Struktura drží **mnoho hodnot pod jedním jménem** a určuje, jak jsou uspořádané: za sebou, bez pořadí, nebo pod klíčem
- Na uspořádání záleží: v šanonu listujete od začátku, v rejstříku skočíte rovnou na heslo; stejná data, jiná rychlost hledání
- Python má čtyři základní struktury vestavěné: `list`, `tuple`, `set`, `dict`; každá se hodí na něco jiného, a o tom je dnešek

# list: seznam hodnot

## list (seznam)

Uspořádaná řada hodnot, do které lze přidávat, z níž lze ubírat a v níž lze hodnoty měnit.

```
>>> smlouvy = ["Kupní smlouva", "Nájemní smlouva", "Smlouva o dílo"]
>>> smlouvy
['Kupní smlouva', 'Nájemní smlouva', 'Smlouva o dílo']
>>> lhuty = [15, 30, 60]
>>> prazdny = []
>>> type(smlouvy)
<class 'list'>
```

- Hranaté závorky, hodnoty oddělené čárkou; prázdný seznam je `[]`
- Záleží na **pořadí**: `[1, 2]` a `[2, 1]` jsou různé seznamy; hodnota se může opakovat
- REPL vypisuje text v jednoduchých uvozovkách; je to stejný text jako ve dvojitých

# list: přístup k prvkům, seznam[i]

- Každý prvek má **index**: pořadové číslo od nuly, stejně jako znaky v textu nebo políčka v paměti
- Záporný index počítá od konce: **-1** je vždy poslední prvek
- **seznam[i]** vrátí prvek; index za koncem skončí chybou **IndexError**
- Výřez **seznam[od:do]** vrátí nový seznam od indexu **od** po **do**, ten už ne

```
smlouvy = ["Kupní", "Nájemní", "0 dílo", "Darovací"]
```

|          | "Kupní" | "Nájemní" | "0 dílo" | "Darovací" |
|----------|---------|-----------|----------|------------|
| index    | 0       | 1         | 2        | 3          |
| od konce | -4      | -3        | -2       | -1         |

zvýrazněno: smlouvy[1:3], tedy od indexu 1 po index 3, ten už ne

# list: index a výřez v praxi

```
>>> smlouvy = ["Kupní", "Nájemní", "0 dílo", "Darovací"]
>>> smlouvy[0]
'Kupní'
>>> smlouvy[-1]
'Darovací'
>>> smlouvy[1:3]                # od indexu 1 po index 3, ten už ne
['Nájemní', '0 dílo']
>>> smlouvy[:2]                 # od začátku
['Kupní', 'Nájemní']
>>> smlouvy[2:]                 # až do konce
['0 dílo', 'Darovací']
>>> smlouvy[4]
IndexError: list index out of range
```

- Poslední platný index je `len(smlouvy) - 1`; `smlouvy[len(smlouvy)]` už je za koncem
- Výřez nikdy nespadne: `smlouvy[2:100]` vrátí, co je, a `smlouvy[10:]` prázdný seznam

# list může obsahovat cokoliv

```
>>> smichany = [17, "Anna", 3.5, True]           # různé typy v jednom seznamu
>>> klienti = ["Anna", 17], ["Petr", 42]]        # seznam seznamů: tabulka
>>> klienti[1]                                    # druhý řádek
['Petr', 42]
>>> klienti[1][0]                                # druhý řádek, první sloupec
'Petr'
>>> len(klienti)                                  # počet řádků, ne buněk
2
```

- Prvkem může být hodnota libovolného typu, i další seznam; tak vznikne tabulka: seznam řádků
- Zvyk: v jednom seznamu držet hodnoty **jednoho typu**; `[17, "Anna", 3.5]` se špatně zpracovává, na záznam s různými položkami je lepší n-tice nebo slovník
- Anotace zapisuje i typ prvků: `smilovy: list[str] = []`, `lhuty: list[int] = [15, 30]`

# Vestavěné funkce pro seznam

| Funkce                                  | Co dělá                  | Příklad                           | Výsledek        |
|-----------------------------------------|--------------------------|-----------------------------------|-----------------|
| <code>len()</code>                      | počet prvků              | <code>len([15, 30, 60])</code>    | 3               |
| <code>sum()</code>                      | součet prvků             | <code>sum([15, 30, 60])</code>    | 105             |
| <code>min()</code> , <code>max()</code> | nejmenší, největší prvek | <code>max([15, 30, 60])</code>    | 60              |
| <code>sorted()</code>                   | nový seřazený seznam     | <code>sorted([30, 15, 60])</code> | [15, 30, 60]    |
| <code>list()</code>                     | převod na seznam         | <code>list("abc")</code>          | ['a', 'b', 'c'] |

- Funkce seznam **nemění**: `sorted()` vrátí nový seznam, původní zůstane, jak byl
- `sorted(lhuty, reverse=True)` řadí od největšího; `min()`, `max()` a `sorted()` umí i texty (podle abecedy)
- Operátor `in` zjistí, zda je hodnota v seznamu: `"Kupní" in smlouvy` dá `bool`

# list lze měnit

```
>>> lhuty = [15, 30, 60]
>>> lhuty[0] = 14           # přepíšeme první prvek
>>> lhuty
[14, 30, 60]
>>> lhuty[1] += 1
>>> lhuty
[14, 31, 60]
>>> lhuty[3] = 90           # přidat přes index nejde
IndexError: list assignment index out of range
>>> slovo = "Právo"
>>> slovo[0] = "p"          # text se takto změnit nedá
TypeError: 'str' object does not support item assignment
```

- Seznam je **měnitelný** (*mutable*): hodnotu na existujícím indexu přepíšeme přiřazením
- Text měnitelný není: metody jako `.upper()` vracejí nový text, původní zůstává
- Nový prvek přidáme metodou `.append()`, hned na dalším slajdu

# Metody seznamu: přidávání a odebírání

Pro `smlouvy = ["Kupní", "Nájemní"]`:

| Metoda                              | Co udělá                           | Výsledek                                              |
|-------------------------------------|------------------------------------|-------------------------------------------------------|
| <code>.append("0 dílo")</code>      | přidá na konec                     | <code>['Kupní', 'Nájemní', '0 dílo']</code>           |
| <code>.insert(0, "Darovací")</code> | vloží na index, zbytek posune      | <code>['Darovací', 'Kupní', 'Nájemní']</code>         |
| <code>.remove("Kupní")</code>       | odebere první výskyt               | <code>['Nájemní']</code>                              |
| <code>.pop()</code>                 | odebere poslední a <b>vrátí ho</b> | <code>'Nájemní'</code> , zbude <code>['Kupní']</code> |
| <code>.pop(0)</code>                | odebere na indexu a vrátí ho       | <code>'Kupní'</code> , zbude <code>['Nájemní']</code> |

- Každý řádek vychází z původního seznamu; `.append()` je zdaleka nejčastější, typicky v cyklu; `.clear()` vyprázdní

# Metody seznamu: přidávání a odebírání v praxi

```
>>> smlouvy = ["Kupní", "Nájemní"]
>>> smlouvy.append("0 dílo")           # přidá na konec
>>> smlouvy
['Kupní', 'Nájemní', '0 dílo']
>>> smlouvy.insert(0, "Darovací")      # vloží na index 0, ostatní se posunou
>>> smlouvy
['Darovací', 'Kupní', 'Nájemní', '0 dílo']
>>> smlouvy.remove("Kupní")            # odebere první výskyt hodnoty
>>> smlouvy.pop()                     # odebere a vrátí poslední prvek
'0 dílo'
>>> smlouvy
['Darovací', 'Nájemní']
```

- Metoda se volá tečkou a seznam **mění na místě**; `.append()` nic nevrací, výsledek nepřisuzujeme
- `smlouvy = smlouvy.append("x")` seznam zničí: uloží do něj `None`, hodnotu „nic“ (více u funkcí)

# Metody seznamu: řazení a hledání

| Metoda                           | Co udělá                             | Z <code>[30, 15, 60, 15]</code> vznikne |
|----------------------------------|--------------------------------------|-----------------------------------------|
| <code>.sort()</code>             | seřadí na místě od nejmenšího        | <code>[15, 15, 30, 60]</code>           |
| <code>.sort(reverse=True)</code> | seřadí na místě od největšího        | <code>[60, 30, 15, 15]</code>           |
| <code>.reverse()</code>          | obráťí pořadí na místě               | <code>[15, 60, 15, 30]</code>           |
| <code>.index(60)</code>          | index prvního výskytu hodnoty        | 2                                       |
| <code>.count(15)</code>          | počet výskytů hodnoty                | 2                                       |
| <code>.extend([90, 120])</code>  | připojí všechny prvky jiného seznamu | <code>[30, 15, 60, 15, 90, 120]</code>  |

- `.index()` chybějící hodnoty skončí `ValueError`; napřed ověřit přes `in`

# sort() a sorted(): nejčastější záměna

```
>>> lhuty = [30, 15, 60]
>>> serazene = lhuty.sort()           # chyba: .sort() řadí na místě a vrací None
>>> print(serazene)
None
>>> lhuty                             # seřazený je původní seznam
[15, 30, 60]
>>> lhuty = [30, 15, 60]
>>> sorted(lhuty)                     # vrátí nový seřazený seznam
[15, 30, 60]
>>> lhuty                             # původní zůstal
[30, 15, 60]
```

- `lhuty.sort()` seznam přeuspořádá a nic nevrátí; `sorted(lhuty)` seznam nechá a vrátí nový
- Pravidlo: metoda mění seznam na místě, funkce vrací nový; `x = lhuty.sort()` je vždy chyba

## Text a seznam: .split() a .join()

```
>>> radek = "Kupní;Nájemní;0 dílo"
>>> smlouvy = radek.split(";")           # text → seznam: rozdělí podle oddělovače
>>> smlouvy
['Kupní', 'Nájemní', '0 dílo']
>>> "Lhůta běží od doručení".split()    # bez argumentu: podle mezer
['Lhůta', 'běží', 'od', 'doručení']
>>> ", ".join(smlouvy)                  # seznam → text: oddělovač je před tečkou
'Kupní, Nájemní, 0 dílo'
```

- Text se chová jako neměnný seznam znaků: index, výřez, `len()`, `in` i `for` fungují stejně
- `.split()` je metoda **textu**, `.join()` metoda **oddělovače**; spojit jde jen texty, čísla napřed `str()`
- Typický vzor: řádek rozdělí `.split()`, zpracovat jako seznam, výsledek slepit `.join()`

# tuple: n-tice

## tuple (n-tice)

Uspořádaná řada hodnot jako seznam, ale **neměnná**. Jednou vytvořená zůstává, jak je.

```
>>> klient = ("Anna", "Nováková", 1998)
>>> klient[0]
'Anna'
>>> klient[2] = 1999
TypeError: 'tuple' object does not support item assignment
>>> jedna = (15,) # jednoprvková n-tice potřebuje čárku; (15) je jen číslo
```

- Kulaté závorky, hodnoty oddělené čárkou; indexy a výřezy fungují stejně jako u seznamu
- Typicky **různé typy**: jeden záznam s pevným významem a pořadím položek (jméno, příjmení, rok)
- Nikdo ji omylem nepřepíše a smí být prvkem množiny i klíčem slovníku (za chvíli)

# tuple: rozbalení a funkce

```
>>> klient = ("Anna", 1998)
>>> jmeno, rok = klient           # rozbalení: každá položka do své proměnné
>>> jmeno
'Anna'
>>> a, b = 1, 2                  # závorky lze vynechat: 1, 2 je n-tice
>>> a, b = b, a                  # prohození hodnot bez pomocné proměnné
>>> a, b
(2, 1)
```

- **Rozbalení** (*unpacking*): počet proměnných musí sedět s počtem položek, jinak `ValueError`
- `len()`, `min()`, `max()`, `sum()`, `sorted()` a `in` fungují jako u seznamu; `sorted()` vrací seznam, `tuple(seznam)` převádí zpět
- Z metod jen `.index()` a `.count()`, protože n-tice se nemění; anotace `klient: tuple[str, int]`

# set: množina

## set (množina)

Hodnoty bez pořadí a bez duplicit. Odpovídá na otázku „je tam, nebo není?“

```
>>> paragrafy = {205, 173, 205, 140}
>>> paragrafy                                # duplicita zmizela, pořadí si Python zvolil sám
{140, 205, 173}
>>> paragrafy[0]                             # bez pořadí není co indexovat
TypeError: 'set' object is not subscriptable
>>> set([1, 2, 2, 3])                         # ze seznamu: pohodlné odstranění duplicit
{1, 2, 3}
```

- Složené závorky; prázdná množina je ale `set()`, protože `{}` je prázdný slovník
- Hodnota je v množině nejvýš jednou; **pořadí neexistuje**, takže ani index, výřez a `.append()`

# set: co do množiny patří a proč je rychlá

```
>>> smichana = {1, "Anna", (2026, 5)}      # čísla, texty, n-tice: ano
>>> len(smichana)
3
>>> {[1, 2], [3]}                          # seznamy: ne
TypeError: cannot use 'list' as a set element (unhashable type: 'list')
```

- Prvkem může být jen **neměnná** hodnota: číslo, text, `bool`, n-tice; seznam, množina ani slovník ne
- Důvod: množina si z každého prvku spočítá číslo (*hash*) a podle něj ho uloží do přihrádky; kdyby se prvek změnil, číslo by neplatilo
- Díky přihrádkám je `in` v množině **velmi rychlé** i při milionu prvků; u seznamu musí Python projít prvek po prvku
- Odtud i „náhodné“ pořadí ve výpisu: je to pořadí přihrádek, ne pořadí vkládání

# set: množinové operace

```
>>> obzaloba = {140, 173, 206}          # paragrafy v obžalobě
>>> rozsudek = {173, 206, 207}         # paragrafy v rozsudku
>>> obzaloba.union(rozsudek)           # sjednocení: aspoň v jedné
{140, 173, 206, 207}
>>> obzaloba.intersection(rozsudek)    # průnik: v obou
{173, 206}
>>> obzaloba.difference(rozsudek)      # rozdíl: v obžalobě, ale ne v rozsudku
{140}
>>> {173, 206}.issubset(obzaloba)      # podmnožina: je vše z první i ve druhé?
True
```

- Výsledek je **nová množina**; `obzaloba` ani `rozsudek` se nemění
- Totéž operátory: `obzaloba | rozsudek`, `obzaloba & rozsudek`, `obzaloba - rozsudek`,  
`{173, 206} <= obzaloba`
- Na pořadí záleží jen u rozdílu: `rozsudek.difference(obzaloba)` dá `{207}`

# set: metody

Pro `mnozina = {140, 173}`:

| Metoda                           | Co udělá                                            | Výsledek                          |
|----------------------------------|-----------------------------------------------------|-----------------------------------|
| <code>.add(206)</code>           | přidá prvek; když už tam je, nic se nestane         | <code>{140, 173, 206}</code>      |
| <code>.remove(140)</code>        | odebere prvek; když tam není, <code>KeyError</code> | <code>{173}</code>                |
| <code>.discard(999)</code>       | odebere prvek; když tam není, nic se nestane        | <code>{140, 173}</code>           |
| <code>.update([206, 207])</code> | přidá všechny prvky ze seznamu nebo jiné množiny    | <code>{140, 173, 206, 207}</code> |

- Vzor „už viděné“ hodnoty: `in` řekne, zda jsme spisovou značku už potkali, `.add()` ji zapíše
- `.pop()` odebere a vrátí libovolný prvek, `.clear()` vyprázdní; `len()`, `min()`, `max()` a `sorted()` fungují jako u seznamu

# frozenset: neměnná množina

## frozenset

Množina, kterou po vytvoření nelze změnit. Umí vše, co `set`, kromě přidávání a odebírání.

```
>>> pevne = frozenset({140, 173, 206})
>>> pevne.add(207)
AttributeError: 'frozenset' object has no attribute 'add'
>>> {frozenset({1, 2}), frozenset({3})} # smí být prvkem množiny, set by nemohl
{frozenset({3}), frozenset({1, 2})}
>>> {{1, 2}, {3}}
TypeError: cannot use 'set' as a set element (unhashable type: 'set')
```

- Vzniká jen funkcí `frozenset()`; vztah k `set` je stejný jako `tuple` k `list`: neměnná verze téhož
- Hodí se jako prvek jiné množiny, klíč slovníku a jako konstanta, kterou nemá nikdo měnit

## dict (slovník)

Dvojice **klíč a hodnota**. Podle klíče hodnotu najdeme jako heslo v rejstříku.

```
>>> tresty = {"krádež": 205, "podvod": 209, "vražda": 140}
>>> tresty["podvod"]           # hodnotu hledáme podle klíče, ne podle indexu
209
>>> tresty["loupež"] = 173      # nový klíč: dvojice se přidá
>>> tresty["krádež"] = 205      # existující klíč: hodnota se přepíše
>>> tresty["zpronevěra"]
KeyError: 'zpronevěra'
```

- Složené závorky, dvojice **klíč: hodnota** oddělené čárkou; prázdný slovník je **{}**
- **Klíč** je neměnný (text, číslo, n-tice) a ve slovníku nejvýš jednou; **hodnota** může být cokoliv
- Dvojice si drží pořadí, v jakém byly vloženy; hledání podle klíče je rychlé jako u množiny

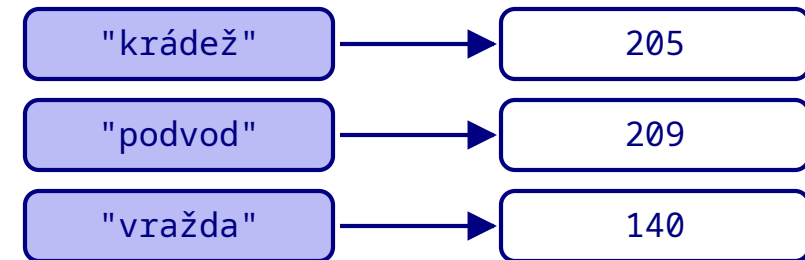
# dict: rejstřík programu

- Rejstřík na konci učebnice: **heslo** → **číslo stránky**. Hledáte heslo, stránku si necháte najít
- Slovník v Pythonu je totéž: klíč je heslo, hodnota je to, co je u něj zapsáno
- Nejjednodušší rejstřík: `rejstrik = {"smlouva": 12, "žaloba": 57}`; pak `rejstrik["žaloba"]` dá `57`
- Stejný vzor: telefonní seznam (jméno → číslo), překladový slovník (slovo → překlad), obrázek vpravo (trestný čin → paragraf)

```
tresty = {"krádež": 205, "podvod": 209, "vražda": 140}
```

klíč: neměnný, nejvýš jednou

hodnota: cokoliv



`tresty["podvod"]` jde po šipce rovnou k 209; index ani pořadí nepotřebuje

# dict: hodnotou může být cokoliv

```
>>> klient = {"jmeno": "Anna", "vek": 17, "smlouvy": ["Kupní", "Nájemní"]}
>>> klient["smlouvy"][0] # hodnota je seznam, dál indexujeme
'Kupní'
>>> klient["smlouvy"].append("Darovací") # a dál na ní voláme metody
>>> len(klient["smlouvy"])
3
>>> klienti = [{"jmeno": "Anna", "vek": 17}, {"jmeno": "Petr", "vek": 42}]
>>> klienti[1]["jmeno"] # druhý záznam, jeho jméno
'Petr'
```

- Záznam o jedné věci: slovník s pevnými klíči ( "jmeno", "vek" ); tabulka: seznam takových slovníků
- Vícestupňový přístup čteme zleva doprava: klienti[1]["jmeno"] je „druhý záznam, z něj jméno“
- Anotace zapisuje typ klíče a hodnoty: tresty: dict[str, int] = {}

# Vestavěné funkce a operátory pro slovník

Pro `tresty = {"krádež": 205, "podvod": 209}`:

| Zápis                           | Co dělá                                  | Výsledek                          |
|---------------------------------|------------------------------------------|-----------------------------------|
| <code>len(tresty)</code>        | počet dvojic                             | 2                                 |
| <code>"podvod" in tresty</code> | je mezi <b>klíči</b> ?                   | True                              |
| <code>209 in tresty</code>      | hodnoty operátor <code>in</code> nehledá | False                             |
| <code>list(tresty)</code>       | seznam klíčů                             | <code>['krádež', 'podvod']</code> |

- Funkce a operátory pracují s **klíči**; `sorted(tresty)` dá seřazené klíče, k hodnotám vedou metody
- `in` před přístupem zabrání `KeyError`: `if "loupež" in tresty:`

# Metody slovníku

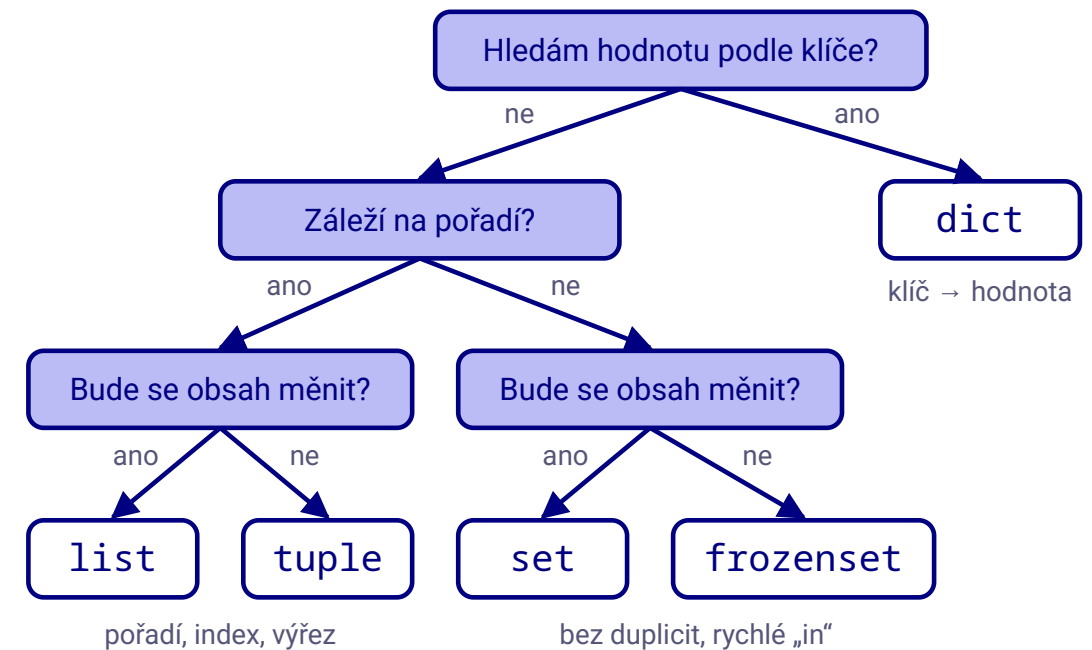
Pro `tresty = {"krádež": 205, "podvod": 209}`:

| Metoda                         | Co udělá                 | Výsledek                                     |
|--------------------------------|--------------------------|----------------------------------------------|
| <code>.get("podvod")</code>    | hodnota podle klíče      | <code>209</code>                             |
| <code>.get("loupež", 0)</code> | náhrada, když klíč chybí | <code>0</code>                               |
| <code>.keys()</code>           | všechny klíče            | <code>dict_keys(['krádež', 'podvod'])</code> |
| <code>.values()</code>         | všechny hodnoty          | <code>dict_values([205, 209])</code>         |

- `.get()` nespadne, když klíč chybí: vrátí `None`, nebo zadanou náhradu
- `.items()` dá dvojice (klíč, hodnota)
- Počítání výskytů, nejčastější vzor: `pocty[slovo] = pocty.get(slovo, 0) + 1`

# Kterou strukturu zvolit

- Potřebuji **pořadí** a přístup podle pozice?  
Seznam; když se obsah nemá měnit, n-tice
- Potřebuji hledat podle **klíče**? Slovník
- Stačí vědět, **zda tam hodnota je**, a duplicity nechci? Množina; neměnná verze je frozenset
- Když nevíte, začněte seznamem: je nejobecnější. Strukturu lze později změnit převodem `list()`, `tuple()`, `set()`, `dict()`



# Přehled: čtyři struktury a jedna navíc

| Struktura | Zápis                | Pořadí | Měnitelná | Duplicity | Použití             |
|-----------|----------------------|--------|-----------|-----------|---------------------|
| list      | [1, 2, 3]            | ano    | ano       | ano       | položky za sebou    |
| tuple     | (1, 2, 3)            | ano    | ne        | ano       | pevný záznam        |
| set       | {1, 2, 3}            | ne     | ano       | ne        | je tam, nebo není   |
| frozenset | frozenset({1, 2, 3}) | ne     | ne        | ne        | neměnná množina     |
| dict      | {"klic": hodnota}    | ano    | ano       | klíče ne  | hledání podle klíče |

- Všechny umí `len()`, operátor `in` a procházení cyklem `for`, ke kterému se dostaneme za chvíli
- Rozhoduje pořadí, měnitelnost a způsob hledání; k tabulce se vraťte, až budete vybírat strukturu pro úkol

# Iterování a cykly

**for: udělej něco s každým prvkem**

# Cyklus for: pro každý prvek seznamu

```
smlouvy = ["Kupní", "Nájemní", "0 dílo"]

for smlouva in smlouvy:          # smlouva postupně nabývá hodnot ze seznamu
    print("Kontroluji:", smlouva)

print("Hotovo")                  # neodsazeno: provede se jednou, až po cyklu
```

```
Kontroluji: Kupní
Kontroluji: Nájemní
Kontroluji: 0 dílo
Hotovo
```

- `for proměnná in seznam:` a odsazený blok; blok proběhne **jednou pro každý prvek**, v pořadí seznamu
- Proměnnou cyklu pojmenujeme sami, nejlépe v jednotném čísle: `for smlouva in smlouvy`
- Oproti `while` nic nepočítáme ani nehlídáme konec: Python ví, kolik prvků seznam má

## for: před cyklem, v cyklu, po cyklu

```
lhuty = [15, 30, 60]

celkem = 0                                # před cyklem: výchozí hodnota
for lhuta in lhuty:
    celkem += lhuta                       # v cyklu: zpracuj aktuální prvek
print("Celkem dnů:", celkem)             # po cyklu: použij výsledek
```

```
dlouhe = []                              # před cyklem: prázdný seznam na výsledky
for lhuta in lhuty:
    if lhuta >= 30:
        dlouhe.append(lhuta)             # vybrané prvky sbíráme do nového seznamu
print(dlouhe)                            # [30, 60]
```

- Tři kroky: **před** cyklem připravit, **v** cyklu zpracovat prvek, **po** cyklu použít výsledek
- Stejný vzor pro součet (ten umí `sum()` za nás), počítadlo, hledání největšího prvku i filtrování

# for funguje nad každou strukturou

```
paragrafy = {205, 173}
for paragraf in paragrafy:           # množina: v pořadí, které si zvolí Python
    print(paragraf)

tresty = {"krádež": 205, "podvod": 209}
for cin in tresty:                   # slovník: po klíčích
    print(cin, tresty[cin])
```

```
205
173
krádež 205
podvod 209
```

- Projít lze text (po znacích), n-tici, množinu i slovník (po klíčích): jsou **iterovatelné**
- Strukturu, kterou právě procházíme, v cyklu **neměníme**; výsledky sbíráme do nové

## range(): čísla od nuly do n

```
for i in range(5):           # 0, 1, 2, 3, 4: pět čísel, od nuly, bez pětky
    print(i)
```

```
>>> list(range(5))          # range čísla nevypíše sám, list() ho rozbalí
[0, 1, 2, 3, 4]
>>> range(5)
range(0, 5)
```

- `range(n)` vyrobí posloupnost `n` čísel od nuly; `n` samo v ní **není**, jako koncový index výřezu
- Použití: „udělej něco pětkrát“ nebo procházení indexů: `for i in range(len(smlouvy))`
- `range` čísla nevyrábí dopředu, ale až když si o ně cyklus řekne; proto se nevypíše jako seznam

# range(): všechny parametry

| Zápis                                 | Znamená                                             | Příklad                      | Čísla     |
|---------------------------------------|-----------------------------------------------------|------------------------------|-----------|
| <code>range(stop)</code>              | od 0 po <code>stop</code> , bez něj                 | <code>range(5)</code>        | 0 1 2 3 4 |
| <code>range(start, stop)</code>       | od <code>start</code> po <code>stop</code>          | <code>range(2, 6)</code>     | 2 3 4 5   |
| <code>range(start, stop, step)</code> | od <code>start</code> po <code>stop</code> s krokem | <code>range(0, 10, 3)</code> | 0 3 6 9   |
| záporný krok                          | sestupně                                            | <code>range(5, 0, -1)</code> | 5 4 3 2 1 |
| prázdný                               | <code>start</code> není před <code>stop</code>      | <code>range(5, 2)</code>     | nic       |

```
>>> list(range(2000, 2027, 5))      # roky po pěti  
[2000, 2005, 2010, 2015, 2020, 2025]
```

- Stejná logika jako výřez `seznam[start:stop:step]`: začátek včetně, konec bez, krok
- Krok nesmí být nula; všechny tři hodnoty jsou `int`, desetinný krok neexistuje

## continue: přeskoč zbytek a jdi na další prvek

```
smlouvy = ["Kupní", "", "Nájemní", "0 dílo"]

for smlouva in smlouvy:
    if smlouva == "":
        continue          # prázdný záznam přeskočíme, cyklus jde dál
    print("Kontroluji:", smlouva)
```

```
Kontroluji: Kupní
Kontroluji: Nájemní
Kontroluji: 0 dílo
```

- `continue` okamžitě **ukončí aktuální průchod**; cyklus pokračuje dalším prvkem
- Hodí se na vyřazení prvků, které nás nezajímají, hned na začátku bloku; zbytek pak nemusí být zanořený v `else`
- Dává smysl jen uvnitř `for` nebo `while`; v obou funguje stejně

## break: ukonči celý cyklus

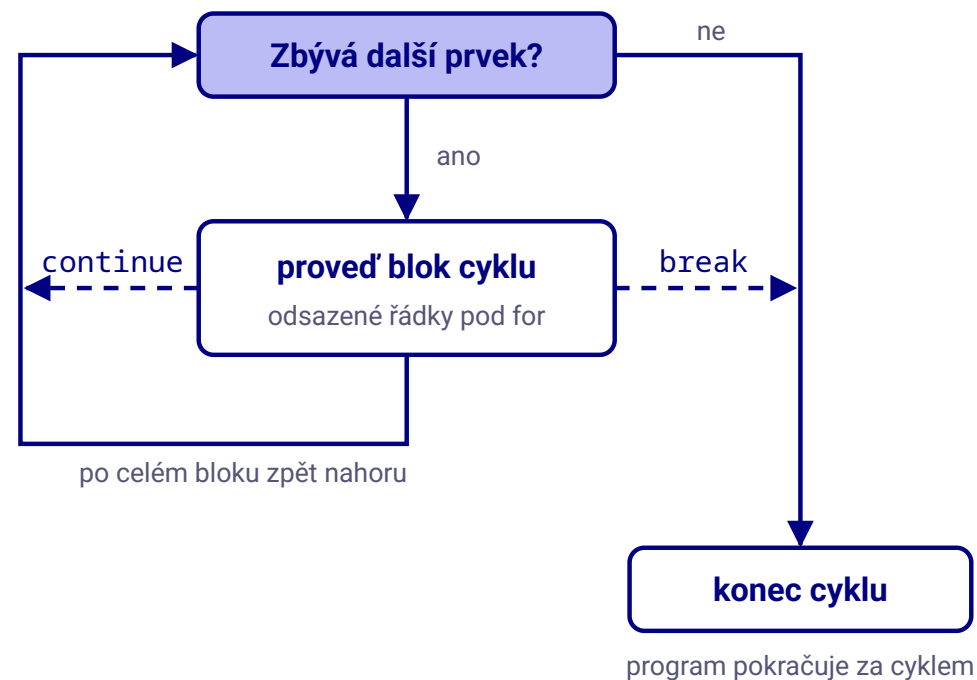
```
smlouvy = ["Kupní", "Nájemní", "0 dílo", "Darovací"]
for smlouva in smlouvy:
    print("Kontroluji:", smlouva)
    if smlouva == "0 dílo":
        break                # ukončí celý cyklus, zbývající prvky se nezpracují
print("Konec")              # pokračuje se prvním řádkem za cyklem
```

```
Kontroluji: Kupní
Kontroluji: Nájemní
Kontroluji: 0 dílo
Konec
```

- `break` **ukončí celý cyklus**; typicky hledání: po první shodě nemá smysl pokračovat
- Vzor `while True:` s `break` uvnitř: opakuj, dokud uživatel nezadá platnou odpověď

# break a continue: tok cyklu

- Běžný průchod: vzít další prvek, provést celý blok, vrátit se pro další
- `continue`: zbytek bloku se přeskočí, cyklus jde rovnou pro další prvek
- `break`: cyklus končí hned, bez ohledu na zbývající prvky
- Když prvky dojdou, cyklus skončí sám a program pokračuje za ním
- Oba příkazy patří k **nejbližšímu** cyklu; ve vnořených cyklech ovlivní jen ten vnitřní



**Děkujeme za pozornost**