Přírodovědecká fakulta UK


# Úvod do programování

**Programovací jazyk Python**

## Dokumentace k programu

# Průnik posloupností

**#102_prunik.py**

Anna Svátková
3. roč., BFGG
Praha, 2021

# Zadání
Program nalezne průnik dvou posloupností reálných čísel (zadání #102).

Ve dvou textových souborech obsahujících posloupnosti reálných čísel budou vyhledány takové hodnoty, které se vyskytují v obou těchto posloupnostech, a budou uložena do nového textového souboru.

# Algoritmus
Získání průniku posloupností je docíleno pomocí postupného procházení seřazených posloupností a porovnávání jejich prvků, kdy v případě, že jsou nalezeny dva stejné prvky, je tato hodnota připojena do nového seznamu společných prvků, pokud v něm doposud není.

# Program
Program načte a validuje vstupní data ze dvou textových souborů a z každé ze dvou posloupností vytvoří seznam hodnot. Pomocí hlavního algoritmu nalezne jejich průnik a seřadí společné hodnoty od nejnižší po nejvyšší. Následně výsledný seznam uloží do nového textového souboru do zdrojové složky.
Data jsou tedy uložena postupně v následujících datových typech:

vstup (IO) → str → list → list → výstup (IO)

Funkce:

- *data_load*: funkce, do které vstupuje název souboru ve zdrojové složce, ve formátu str, včetně přípony. S využitím modulu *os* je zjištěno, jestli soubor s daným názvem ve složce existuje, jestli není prázdný nebo příliš velký (limitem je 50 MB), jestli je k němu povolen přístup čtení a jestli je čitelným (nebinárním...) v rámci Unicode. Pokud některá z podmínek není splněna, program vypíše chybovou hlášku a ukončí se. Pokud problém nenastane, funkce načte data ze souboru, převede je na datový typ str (řetězec) a odstraní bílé znaky z obou konců řetězce. Následně zavolá vnořenou funkci *structure_check* (viz níže). Funkce vrací posloupnost s validovanou strukturou jako str.

- *structure_check*: vnořená funkce volaná uvnitř funkce *data_load*. Vstupem je řetězec obsahující text ze vstupního souboru, s odstraněnými bílými znaky na obou koncích. (Dále také vstupuje název souboru (str) kvůli výpisu v chybové hlášce.) Funkce využívá regulárního výrazu, definovaného v proměnné ‚structure‘, který popisuje strukturu vstupu tak, aby odpovídala posloupnosti (kladných nebo záporných) reálných čísel oddělovaných jednou mezerou, s tečkou jako desetinným znaménkem. Vzorek (pattern) regulárního výrazu popisuje validní strukturu celého vstupního řetězce. Detailní rozbor vzorku je k nalezení v sekci Ošetření nevalidních vstupů. Vstupní řetězec je se vzorkem porovnán, a pokud mu neodpovídá, program vypíše chybovou hlášku a ukončí se (přeruší tak i funkci *data_load*). V případě, že struktura vstupního řetězce je validní, funkce program nepřeruší, ani nic nevrací (nechá dále pokračovat funkci *data_load*).

Po získání obou posloupností pomocí funkce *data_load* je v hlavní části programu z každé posloupnosti (str) vytvořen seznam (list) jednotlivých reálných čísel pomocí iterování přes řetězec rozdělený funkcí *split* (viz níže). Zároveň je také v tomto kroku každé získané číslo převedeno ze z řetězce (str) na datový typ čísla s plovoucím desetinným znakem (tečkou) – float. Výsledkem jsou tedy dva seznamy (list) čísel (float).

- *split*: vestavěná funkce, která rozdělí vstupní řetězec (str) do seznamu podle oddělovačů. Výchozím oddělovačem je jakýkoliv bílý znak, v tomto programu je ale jako oddělovač zvolena jedna mezera. Vrací seznam (list).

- *inters*: do funkce vstupují dva předešle vytvořené seznamy prvků. V případě tohoto programu jde vždy o prvky, které jsou reálnými čísly ve formátu float. Funkce vyžaduje, aby do ní vstupovaly seznamy seřaditelných a porovnatelných prvků. Položky vstupních seznamů jsou zde pomocí funkce *sort* (viz níže) seřazeny vzestupně. Pro každý seznam je vytvořen iterátor. Následně je započat cyklus, ve kterém je vzata vždy hodnota z jednoho seznamu i ze druhého, ty na které ukazují iterátory vždy daného seznamu. Hodnoty jsou porovnány, pokud se rovnají, pak je přidána tato hodnota do nově vznikajícího seznamu a hodnoty obou iterátorů seznamů se zvýší o 1. Pokud ale některá z hodnot menší než ta druhá, je pak vždy o 1 zvýšena jen hodnota iterátoru právě tohoto seznamu, čímž se v následující iteraci bude porovnávat místo této s hodnotou v tomto seznamu následující (tedy vyšší hodnota reálného čísla). Tento cyklus běží, dokud není dosaženo posledních prvků seznamů (včetně). Funkce zamezuje duplicitnímu zápisu společných prvků seznamů pomocí podmínky, neukládá tedy informaci o tom, kolikrát se společný prvek v prvním ze seznamů vyskytuje. Odebráním podmínky unikátnosti ve výsledném seznamu by se dalo toto upravit. Výstupem funkce je tedy seznam (list) obsahující prvky společné oběma vstupním seznamům (nezávisle na počtu těchto prvků v rámci jednotlivých seznamů).

- *sort*: vestavěná funkce, která na rozdíl od podobné funkce *sorted* nic nevrací, avšak mění (seřazuje) samotný vstupní seznam (list), na který je volaná. V programu je funkce nastavena na výchozí pořadí, prvky seznamu jsou tedy seřazeny vzestupně. V současném stavu programu je umístění této funkce na konci programu nadbytečné (hodnoty jsou tímto způsobem seřazeny již od funkce *inters*. Pokud by byl změněn systém výpočtu (např. nahrazením za alternativu), mohla by být nutná pro zachování tohoto pořadí výsledků.

- *file_out*: vytvoří ve zdrojové složce soubor ve formátu TXT, který obsahuje vysvětlující informaci o výstupních datech, následovanou výstupními daty. V případě tohoto programu do funkce vstupuje seznam (list) hodnot společných pro oba globální vstupy programu (vytvořen funkcí *inters* a seřazen funkcí *sort*), který je vložen za informaci do výstupního souboru. Funkce zapíše do výstupního souboru i data jiných datových typů, pro pravdivost obsahu výstupního souboru by se však musela upravit vypisovaná informace o výstupních datech.

# Vstupní data

Vstupními daty jsou dva textové soubory. Tyto soubory musí být pojmenovány ‚values1‘ a ‚values2‘. Dále musí být ve formátu TXT a musí se nacházet ve zdrojovém adresáři programu, tedy ve stejném adresáři, ve kterém se nachází program. Další povinné parametry vstupů jsou pro oba soubory rovněž společné.

Každý vstupní soubor musí obsahovat posloupnost reálných čísel oddělených přesně jednou mezerou. Čísla nesmí být na více řádcích, ani oddělena čímkoliv jiným, než mezerou. Přebytečné bílé znaky (mezery, nové řádky apod.) na začátku a na konci posloupnosti (před prvním číslem a za posledním číslem) nehrají roli. V případě desetinných čísel musí být vždy použita desetinná tečka (nikoliv čárka). Záporným číslům musí předcházet jeden znak „-“ (pomlčka, minus). Kladná čísla nesmí být označena znakem „+“. Posloupnost nesmí obsahovat žádné jiné znaky, než jsou číslice 0-9, desetinná tečka, pomlčka (minus; ASCII 45) a mezery.

V souhrnu musí obsah každého vstupního souboru odpovídat struktuře podle následujícího popisu:

- minus v případě záporného čísla;
- reálné číslo: celé číslo (libovolný počet číslic) nebo desetinné číslo (libovolný počet číslic, desetinná tečka a opět libovolný počet číslic);
- oddělovač = jedna mezera

→ a poté další kladné/záporné reálné číslo, atd.
Posloupnost může rovněž obsahovat pouze jediný prvek (pak není potřeba žádných mezer pro oddělení). Program zpracuje vstupní data se znaky podle kódování UTF-8.

Příklad validního vstupu:
15 7.67 -26 485 -321.24 98 -0.756 3

Pro otestování programu lze použít přiložená ukázková vstupní data.

# Výstupní data

Výstupem z programu je textový soubor s názvem ‚sequences_intersection' ve formátu TXT, který bude vytvořen ve zdrojovém adresáři programu (viz Vstupní data). Soubor obsahuje společné hodnoty vstupních posloupností, a také informaci, že jde právě o tyto hodnoty (na jednom řádku). Výsledky jsou vypsány v hranatých závorkách, v případě více než jedné společné hodnoty jsou oddělovány čárkou a mezerou. Společné hodnoty jsou seřazeny vzestupně, tedy od nejmenší po největší zleva doprava. Pokud jsou mezi výsledky celá čísla, jsou vypsána jako desetinná čísla s nulou za desetinnou tečkou. Program vytváří výstupní data se znaky podle kódování UTF-8.

Příklad výstupu:
The intersection of sequences from the input files are the following values: [15.0 -26.0 98.0 -0.756]

# Ošetření nevalidních vstupů

V programu jsou dva hlavní druhy ošetření nevalidních vstupů – ošetření řešící souborové chyby vstupů a ošetření řešící strukturní chyby vstupů. Chybové hlášky obsahují informaci o tom, ve kterém ze vstupních souborů se vyskytla chyba (za využití f-string). Nejprve jsou pomocí modulu *os* zjištěny případné souborové chyby:

- zda se ve zdrojové složce nachází soubor s daným názvem: funkce *path.exists* z modulu *os*
  - při neexistenci souboru program vypíše chybovou hlášku a ukončí se

- zda soubor s daným názvem není prázdný (jeho velikost není nulová) nebo větší než 50 MB: funkce *path.getsize* z modulu *os*
  - při nulové velikosti souboru nebo větší než je limit program vypíše chybovou hlášku a ukončí se

- zda je k souboru povolen přístup ke čtení (mimo modul *os*): ošetření výjimky pomocí *try* & *except* pro PermissionError; pod *try* je voláno otevření a čtení dat ze souboru, *except* zabrání ukončení programu chybou
  - když není právo pro čtení souboru, program místo chyby vypíše chybovou hlášku a ukončí se
- zda je soubor čitelný v rámci Unicode – součástí předchozího *try* s vlastním *except* pro UnicodeDecodeError (viz předchozí bod)
  - když není soubor čitelný, program místo chyby vypíše chybovou hlášku a ukončí se

Poté je zavolána funkce *structure_check*, do které vstupují data načtená ze souboru, ve formátu řetězce. V programu je pro kontrolu validní struktury použito regulárního výrazu, který popisuje jakékoliv reálné číslo, příp. reálná čísla a jejich oddělovače (mezery). Vzorek, se kterým se celý řetězec porovnává, je následující:

`^(-?\d+(\.\d+)?)( -?\d+(\.\d+)?)*$`

- `^` na začátku zajišťuje to, že je shoda hledána vždy od začátku řetězce (musí se shodovat začátek vzorku se začátkem řetězce).
- `$` na konci zajišťuje to, že konec řetězce musí odpovídat konci vzorku (analogicky k ^).

(Tyto dva znaky jsou použity za účelem, aby pro potvrzení validity řetězce musel vždy vzorku odpovídat celý řetězec od začátku do konce.)

- `-?` určuje, že se na začátku čísla může nebo nemusí vyskytovat jedno minus
- `\d+` popisuje celou část čísla, vyjádřenou jednou nebo více číslicemi (0-9)

Následuje podskupina, která zde (jak vyjadřuje znak otazníku) buď nemusí být vůbec (v případě celého čísla), nebo je zde přesně jednou, a to v případě desetinného čísla, kdy vyjadřuje jeho desetinnou část:

- `\.` vyjadřuje desetinnou tečku
- `\d+` popisuje desetinnou část čísla, vyjádřenou opět jednou nebo více číslicemi (0-9)

Poslední částí vzorku je skupina, která je totožná s předchozí částí popisující kompletní strukturu reálného čísla, ale s tím rozdílem, že je na jejím začátku přidaná mezera jakožto oddělovač. Za touto skupinou je znak * (hvězdička), který určuje, že se zde skupina nemusí nacházet vůbec (v případě, kdy str obsahuje pouze jediné reálné číslo), nebo v jakémkoliv (neomez.) počtu.

V měřítku reálných čísel tedy struktura vypadá následovně:

(<reálné číslo>)(<mezera><reálné číslo>),

s tím, že druhá skupina se vyskytuje přesně (n-1)krát, kdy n je počet reálných čísel v posloupnosti.

Pokud tedy struktura vstupního řetězce odpovídá výše vysvětlenému vzorku, program běží nepřerušen dále. Pokud struktura v jakémkoliv aspektu neodpovídá, je program přerušen a vypíše chybovou hlášku, upozorňující na nevalidní strukturu daného vstupu.

# Alternativní řešení

Souborové chyby:

Souborové chyby by mohly být také ošetřeny například pomocí různých módů funkce *os.access* (módy: *os.F_OK*: existence souboru, *os.R_OK*: právo čtení apod.), nebo *os.path.isfile(<nazev.txt>)* (existence souboru) a také *os.stat(<nazev.txt>).st_size == 0* (velikost souboru = zda není prázdný). Existují i další způsoby, jak tyto informace o souboru zjistit. V případě souboru mimo zdrojovou složku je potřeba doplnit jeho název s příponou kompletní cestou k tomuto souboru.

Bylo by také možné dát souborové podmínky společně do jednoho podmiňovacího příkazu (řádku kódu) se třemi podmínkami, kdy pro pokračování musí být splněny všechny. Pak by byl zkrácen kód, ale také by byla jen jedna společná chybová hláška pro všechny tři chyby, zatímco odděleně lze lépe uživatelsky poznat, kde se problém týkající se souboru nachází.

Validace struktury:

I pro kontrolu správné struktury lze využít jiné řešení. Je například možné rozdělit vlastnosti struktury na jednotlivé podmínky. Příklady dalších způsobů řešení těchto podmínek řetězce:

- musí obsahovat minimálně jednu číslici
  - *if <číslice (or)> not in <řetězec>* → ukončení programu
  - využití regulárního výrazu; neshoda s žádným z prvků vzorku → ukončení programu (viz kód alternativní funkce níže)

- nesmí obsahovat dvě a více mezer za sebou (bílé znaky z konců jsou odstraněny již dříve)
  - *if '<dvě mezery>' in <řetězec>* → ukončení programu

- nesmí obsahovat nepodporované znaky – jen mezery, pomlčky, tečky a číslice
  - využití regulárního výrazu; shoda s jakýmkoliv z prvků vzorku → ukončení programu (viz kód alternativní funkce níže)

- každá hodnota posloupnosti musí být reálné číslo (int nebo float, ale float může zastřešit hodnoty obou těchto datových typů)

        - nevýhodou je to, že je pro ověření této vlastnosti dat nutné před tím data rozdělit do seznamu → takže z hlediska sestavení programu se rozdělení (split) seznamu buď musí uskutečnit již v rámci validační funkce, nebo se naopak ověření správné struktury uskuteční až při zpracování dat
    - využití regulárního výrazu; srovnávání každého prvku se vzorkem popisujícím reálné číslo (^-?\d+(\.\d+)?$); neshoda s minimálně jedním z prvků → ukončení programu
    - *try* & *except* – ‚vyzkoušení' na postupu, který vygeneruje chybu v případě, že půjde o prvek, který nelze daným postupem zpracovat; chyba je ale podchycena výjimkou → rovněž vede k ukončení programu (nikoliv k jeho pádu)
      Např. v alternativní funkci níže je použito převodu jednotlivých prvků seznamu (list) vzniklého rozdělením řetězce (str) na menší řetězce, které by měly odpovídat vždy jednomu reálnému číslu, na datový typ float. Pokud struktura prvku neodpovídá datovému typu float (např. je zde ‚-1.5-..' místo ‚-1.5'), nezdaří se převod a vzniká chyba ‚ValueError', která je poté podchycena. Níže uvedený jednořádkový zápis rozdělení seznamu je velmi úsporný a postaven tak, že není potřeba, aby byl vytvořený seznam použit, a stal se tak výstupem funkce, která by jinak výstup neměla (což by byla určitá komplikace). Zároveň i přes částečné zpracovávání dat ve validační funkci, které je tím způsobeno, zde toto nezabírá příliš prostoru a nijak výrazněji nevadí.

Nevýhodou rozdělení validace struktury na jednotlivé podmínky je zvýšení množství potřebného kódu (pokud se opět nezapíše do jedné vícenásobné podmínky). Výhodou je naopak pro uživatele jednoduché rozpoznání, kde se vyskytla chyba (chybové hlášky jsou zvlášť). V programu lze jednoduše nahradit funkci aplikující jeden regulární výraz na celý řetězec funkcí, která jednotlivé podmínky odděluje. Stačí nahradit funkci *structure_check* následující funkcí (příklad aplikace):

```
def structure_check(string):
    complement_ok_chars = r"[^\-0-9\. ]"
    digits = r"[0-9]+"
    if '  ' in values:
        print('Invalid separation by spaces – too many spaces.')
        exit()
    elif re.match(complement_ok_chars, values) is not None:
        print('Invalid input – contains unsupported characters.')
        exit()
    elif re.match(digits, values) is None:
        print('Invalid input – does not contain digits.')
        exit()
    else:
        try:
            [float(item) for item in values.split(' ')]
        except ValueError:
            print("Input does not have valid structure.")
            exit()
```

Za použití této funkce program akceptuje i nekompletní hodnoty jako např. ‚5.' nebo ‚.3', které bere jako ‚5.0', resp. ‚0.3'. Ponechání tohoto záleží na požadavcích a případně vyžaduje další úpravy kódu.

Ve výsledku bylo tedy použití regulárního výrazu na celou délku vstupního řetězce prospěšné pro délku kódu a jeho vnitřní strukturu, ale uživatel nezjistí v případě strukturní chyby tak podrobně příčinu této chyby.

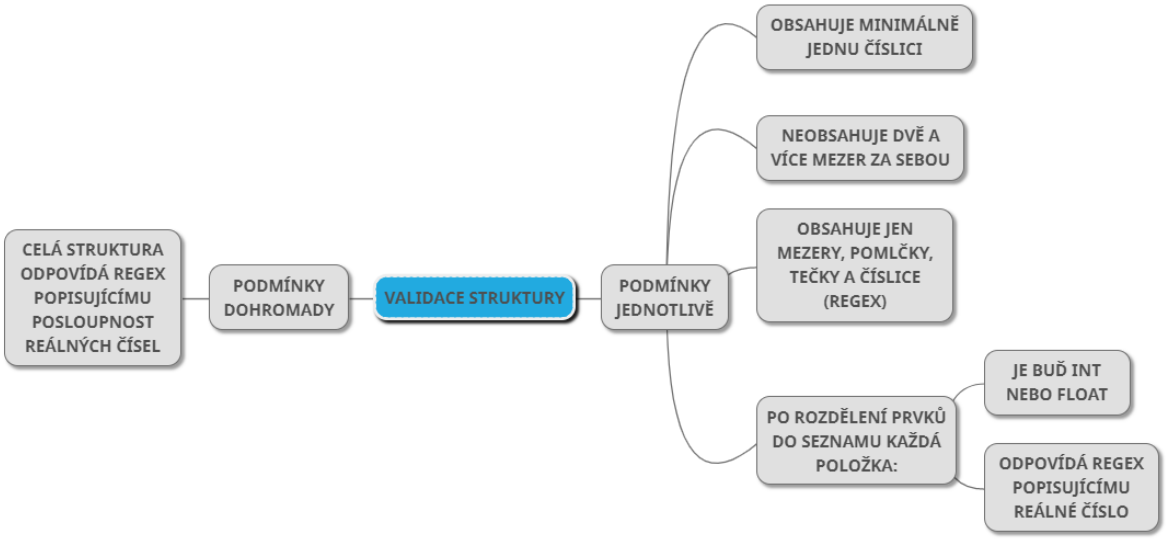


*Schéma 1: Příklady možností validace struktury*

Průnik:
Dalším možným řešením hledání průniku posloupností by bylo například využitím setů v jedné z následujících možností:

- použití funkce *intersection* na dva nebo více setů prvků:
  - *set_A.intersection(set_B,set_C)* → vrací prvky společné pro všechny sety
  - Po rozdělení každého vstupu do seznamu by byla možná následující aplikace:
    *set.intersection(set(list_A), set(list_B),set(list_C)*
    → výstupem je set obsahující průnik dvou seznamů (možné převést na seznam).
- použití operátoru *&*:
  - *set_A & set_B & set_C* → vrací prvky společné pro všechny sety
  - Po rozdělení každého vstupu do seznamu by byla možná následující aplikace:
    *set(list_A) & set(list_B) & set(list_C)*
- V případě, že neexistují žádné prvky společné pro všechny zúčastněné sety (včetně případů, kdy existují společné prvky pro některé ze setů, ale ne pro všechny), místo hodnot vrací tyto aplikace následující:
  - *set()*

Jinou možností je získání průniku posloupností iterováním přes seznam prvků jedné z posloupností, kdy v případě, že se prvek vyskytuje i v seznamu z druhé posloupnosti, je připojen do nového seznamu společných prvků (pokud v něm doposud není). Tento způsob je výrazně pomalejší (vícenásobná iterace přes posloupnosti). Pro jeho aplikaci je potřeba nahradit funkci *inters* za následující funkci:

```
def inters(list_values1,list_values2):
    list_inters = []
    for number in list_values1:
        if number in list_values2:
            if number not in list_inters:
                list_inters.append(number)
    return(list_inters)
```

Tato funkce nevyžaduje konkrétní vlastnosti prvků. Mohla by tedy být použita na zjištění průniku dvou seznamů obsahujících i jiné datové typy (nahrazující proměnnou ‚number‘). Funkce zamezuje duplicitnímu zápisu společných prvků seznamů, neukládá tedy informaci o tom, kolikrát se společný prvek v prvním ze seznamů vyskytuje. Odebráním podmínky unikátnosti ve výsledném seznamu by se dalo toto upravit, v tom případě by ale záleželo na pořadí vstupujících seznamů (ve výsledném seznamu by bylo tolik společných prvků, kolik jich je v prvním ze seznamů, i když by jich bylo ve druhém seznamu více). Výstupem funkce je tedy seznam (list) obsahující prvky společné oběma vstupním seznamům (nezávisle na počtu těchto prvků v rámci jednotlivých seznamů).

## Další možnosti vývoje

Návrhy na možná vylepšení/rozšíření programu:
Výstupní chybové hlášky sice obsahují informaci o tom, ve kterém ze vstupních souborů se vyskytla chyba (f-string), mohly by případně informovat i o tom, kde v souboru se tato chyba (chyby) nachází. Pokud by to bylo z uživatelské strany potřeba, je možné rozlišit více druhů chybových hlášek. Dále by mohla být umožněna uživatelská interakce s programem (mimo přípravu vstupních souborů), jako například: zadávání vlastního názvu vstupních či výstupního souboru, zadávání vlastní cesty k vstupnímu souboru nebo výběr umístění pro výstupní soubor (v případě, že by nebyly ve stejném adresáři jako program); dále pak další nastavitelné parametry, jako určení podle jakých oddělovačů se mají posloupnosti rozdělit, podle jakých pravidel řadit výstup npod.
Dále by mohl program umět zpracovat i posloupnost s entery, tabulátory, středníky, čárkami apod. jako oddělovači – buď v závislosti na uživatelském výběru, nebo automaticky detekovat pravděpodobné oddělovače, nebo nahradit různé oddělovače pouze za jeden určitý znak (např. mezeru). Jednou z možností je také třeba další přizpůsobení výpisu výsledků (např. vypisování celých čísel bez nulových desetinných míst).