Přírodovědecká fakulta UK

Úvod do programování

Programovací jazyk Python

Dokumentace k programu

Výpočet četnosti

#4_cetnost.py

Anna Svátková
3. roč., BFGG
Praha, 2021

Zadání
Program vypočítá četnosti jednotlivých znaků ve vstupním textu (zadání #4).

Z textového souboru budou načtena vstupní data a následně budou pro všechny v textu se vyskytující znaky spočítány jejich celkové (absolutní) četnosti, které se pak uloží do nového textového souboru.

Algoritmus
Získání četností znaků je docíleno pomocí iterování přes znaky řetězce. Pro tento účel je vytvořen nový slovník, do kterého se vždy zaznamená znak jako klíč, pokud tam ještě není. Každý první výskyt tedy vytvoří pro daný znak klíč s hodnotou 1. Každý další výskyt stejného znaku tutuo hodnotu zvýší o 1.

Program
Program načte a validuje vstupní data z textového souboru, určeného uživatelem. Pomocí hlavního algoritmu spočítá četnosti jednotlivých znaků a uloží názvy znaků a hodnoty jejich četnosti do nového textového souboru do zdrojové složky.
Data jsou tedy uložena postupně v následujících datových typech:

vstup (IO) → str → dict → str → výstup (IO)

Funkce:

- *data_load*: funkce, do které vstupuje název souboru (ve formátu str) ve zdrojové složce, zadaný uživatelem, 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 a poté je převede na datový typ str (řetězec). Funkce vrací obsah vstupního souboru jako str.

- *char_frequency*: funkce, do které vstupuje řetězec vytvořený funkcí *data_load* z obsahu globálního vstupu. I když je tento program sestaven tak, že do této funkce vstupuje datový typ str, funkce bude správně zpracovávat i data, která budou ve formátu seznamu (list), čehož je možné využít při některých obměnách kódu (viz sekce Alternativní řešení). Ve funkci je vytvořen prázdný slovník (dict) s názvem *frequency_dict*, následně pak probíhájí iterace přes vstupní řetězec (případně seznam znaků). Při každé iteraci je rozhodováno v rámci podmínky následující:
    - není-li ve slovníku záznam znaku (při prvním výskytu znaku), je v něm vytvořen. Klíčem je znak a do hodnoty je přiřazeno číslo 1. Příklad záznamu: 'a': 1
    - je-li již ve slovníku záznam daného znaku, je hodnota pod klíčem tohoto znaku zvětšena o 1.

  Funkce vrací slovník (dict) se záznamy, kde se pod klíči znaků (str) nacházejí hodnoty četnosti těchto znaků (int).

- *file_output*: funkce, do které vstupuje slovník (dict) vytvořený ve funkci *char_frequency* a obsahující údaje o četnosti jednotlivých znaků. Funkce vytvoří ve zdrojové složce soubor s názvem ‚cetnosti_znaku‘ ve formátu TXT. Do tohoto souboru je vždy vypsáno o jaký znak se jedná a kolik jich ve vstupním souboru programu je. Např.: Počet 'a' v souboru je 12.

Současný stav programu zahrnuje při počítání četnosti i bílé znaky – tedy mezery, odřádkování (enter) apod. Tabulátor je počítán jako 4 mezery.

Vstupní data
Vstupními daty je jeden textový soubor, jehož název musí uživatel zadat v přesném znění včetně přípony. Zároveň se také musí nacházet ve stejném adresáři jako tento program. Vstupní soubor může obsahovat jakékoliv znaky, tedy text, čísla a všechny další znaky (včetmě bílých znaků), v rámci kódování UTF-8.

Příklad validního vstupu:
Sveřepí šakali zavile vyli na bílý měsíc.

Pro otestování programu lze použít přiložená ukázková vstupní data.

Výstupní data
Výstupem z programu je textový soubor s názvem ‚char_frequencies‘ ve formátu TXT, který bude vytvořen ve zdrojovém adresáři programu (tam, kde se nachází program i jeho vstupní data). Tento soubor pak obsahuje výpis znaků, které se vyskytují ve vstupním souboru, spolu s k nim patřícími četnostmi – tedy údaji o tom, kolikrát se zde daný znak nachází. Výpis informace o četnosti každého znaku je zaznamenán vždy na vlastním řádku. Výsledky sčítání jsou seřazeny podle toho, v jakém pořadí se nachází ve vstupním souboru. Pro každý unikátní znak je vypsán jeden záznam s celkovou (absolutní) hodnotou četnosti tohoto znaku v souboru. Program vytváří výstupní data v kódování UTF-8.

Příklad výstupu:
The number of 'c' in the file is 4.
The number of ',' in the file is 8.
The number of '&' in the file is 1.

Ošetření nevalidních vstupů
V programu jsou ošetřeny souborové chyby, týkající se vstupního souboru. Chybové hlášky obsahují informaci o tom, ve kterém ze vstupních souborů se vyskytla chyba (za využití f-string).
Ošetření výjimek je zajištěno pomocí modulu *os* v následujících případech:

- pokud se ve zdrojové složce nenachází soubor s daným názvem: funkce *path.exists* z modulu *os*
  - při neexistenci souboru program vypíše chybovou hlášku a ukončí se
- pokud je soubor s daným názvem prázdný (jeho velikost je nulová) nebo velikost větší než 50 MB: funkce *path.getsize* z modulu *os*
  - při velikosti souboru nulové nebo nadlimitní program vypíše chyb. hlášku a ukončí se
- pokud není 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
- pokud není 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

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.

Také lze omezit formát vstupu. Příklad pro omezení pouze na formát TXT:

```
    if not file_name.lower().endswith('.txt'):
        print(f"The input file {file_name} is not in the correct format.")
        exit()
```

Četnost:
Řešení výpočtu četností aplikované v tomto programu sčítá i bílé znaky. Jinou možností, jak četnosti spočítat, a při tom bílé znaky nebrat v úvahu, je například postup s využitím seznamu. Toto lze do kódu implementovat třeba přidáním funkce, která dělí vstupní řetězec na seznam. Například zařadit tuto funkci (byla součástí předchozí verze tohoto programu):

```
def split_char(string):
    list_chars = []
    list_items = string.split()
    for item in list_items:
        for char in item:
            list_chars.append(char)
    return(list_chars)
```

Seznam znaků (list) ze vstupního řetězce (str) je získán pomocí iterování přes tento řetězec, rozdělený funkcí split bez určetní oddělovače. Tímto způsobem proběhne dělení podle bílých znaků, a tyto znaky pak nebudou součástí výstupního seznamu. Následně jsou všechny takto vzniklé prvky seznamu (u běžného textu zpravidla řetězce tvořené několika znaky) po jednotlivých znacích přidány do nového seznamu – seznamu všech znaků v souboru (v pořadí v jakém jsou v souboru, včetně vícenásobných výskytů).

V programu je současně s tím potřeba nahradit také tento řádek:

```
char_frequency = char_frequency(text)
```

těmito řádky:

```
all_chars = split_char(text)
char_frequency = char_frequency(all_chars)
```

Jde o přidání mezikroku tvorby seznamu. Funkce *char_frequency* není vázaná na to, jestli do ní bude vstupovat řetězec nebo seznam.

I výše zmíněná verze využívající seznam by se dala přizpůsobit tak, aby počítala i bílé znaky. Tato obměna by mohla být využita, pokud by bylo potřeba sčítání bílých znaků a zároveň i vznik seznamu všech znaků ze souboru (např. pro další zpracování tohoto seznamu). Tato úprava spočívá ve využití vestavěné funkce *enumerate*. Je pro to potřeba nahradit vnější cyklus ve výše zmíněné funkci *rozdeleni* (za vytvořením prázdného seznamu ‚list_chars‘) tímto cyklem:

```
for count,element in enumerate(string):
    list_chars.append(element)
```

Funkce *enumerate* očísluje jednotlivé položky iterovatelného argumentu, přes který běží cyklus (přiřadí jim pořadí). Tato funkce vrací dvojice prvků, na první pozici je pořadí a na druhé položka argumentu, které toto pořadí patří. Vypsáním dvou proměnných v hlavičce cyklu je možné získat

prvky n-tice zvlášť a připojit položku dané iterace na konec nově vznikajícího seznamu. Právě tak by bylo ekvivalentní i použití *list_chars.append(string[count])*, které by rovněž připojilo položku daného pořadí na pozici vstupního řetězce. Do funkce *enumerate* musí pro vznik seznamu všech znaků vstupovat právě řetězec, pokud by byl jejím argumentem např. seznam. Cyklus by pak probíhal přes jeho celé položky, i když by byly tvořeny více znaky (a do *list_chars* by nebyly připojovány jednotlivé znaky). Tabulátor je počítán opět jako 4 mezery.

Výpočet četnosti má mimo výše zmíněné obměny také další alternativní řešení. Jedním z nich je použití funkce C*ounter* z modulu *collections*. Po importu funkce z knihovny by syntaxe tohoto řešení vypadala následovně:

Counter(řetězec)

Výstupem této funkce je pak kolekce obsahující slovníkové záznamy – klíče a jejich hodnoty, které obsahují vždy četnost daného klíče. Záznamy jsou zde za sebou od nejpočetnějšího k nejméně početnému. Výstup může vypadat např. takto:

Counter({'n': 6, 'b': 4, 'a': 2, 'c': 1})

Výhodou tohoto postupu je v případě potřeby dotazovat se na jednotlivé četnosti znaků. Při dotazu na znak, který se ve vstupním řetězci vůbec nevyskytuje totiž program není ukončen kvůli chybě klíče (který není nalezen), ale vrací nulu. Sčítány jsou i bílé znaky, ale např. nový řádek je pak vypsán jako '\n'.

Výstup:

Ačkoliv jsou výsledky sčítání ve výstupním souboru seřazeny podle toho, v jakém pořadí se nachází ve vstupním souboru, jsou i další pravidla, podle kterých se může výpis řadit. Například:

- řazení klíčů podle ASCII kódu:

```
with open('char_frequencies.txt', mode='w', encoding='utf-8') as output:
    s_frequency_dict_out = dict(sorted(frequency_dict_out.items()))
    for char_key,char_count in s_frequency_dict_out.items():
        count = f"The number of '{char_key}' in the file is {char_count}."
        print(count, file=output)
```

- řazení hodnot podle početnosti:

```
with open('char_frequencies.txt', mode='w', encoding='utf-8') as output:
    s_frequency_dict_out = dict(sorted(frequency_dict_out.items(), key=lambda item:
  item[1], reverse=True))
    for char_key,char_count in s_frequency_dict_out.items():
        count = f"The number of '{char_key}' in the file is {char_count}."
        print(count, file=output)
```

  - volitelný argument ‚reverse‘ zde určuje směr řazení:
    - *reverse=True* znamená sestupné řazení (od nejčetnějšího, viz výše)
    - *reverse=False* nebo bez tohoto argumentu znamená vzestupné řazení

Další možnosti vývoje

Návrhy na možná vylepšení/rozšíření programu:

V sekci Alternativní řešení je zmíněna řada úprav, kterými je možné program přizpůsobit. Praktickou možností jsou úpravy pořadí výstupu, který lze kromě pořadí ze souboru řadit např. také podle četnosti, nebo podle kódu ASCII/abecedního pořadí. Je také možné podle potřeby s využitím seznamu nepočítat bílé znaky, nebo seznam vytvořit a přesto bílé znaky počítat.

Mimo to se nabízí další rozšíření, jako např. výpis jen nejpočetnějšího/nejméně početného, dále pak lze umožnit, aby byl program uživatelsky interaktivní. To může zahrnovat např.:

- výběr jen určité množiny znaků pro výpočet četností
- vynechání určité množiny znaků pro výpočet četností
- možnost nastavení vlastního názvu výstupního souboru
- možnost nastavení vlastní cesty ke vstupnímu souboru
- možnost nastavení vlastního umístění výstupního souboru