Zadania (05.12.2025)

Jeszcze kilka zadań do poćwiczenia działania na plikach i listach, a także zbiorach (które omawialiśmy bardzo pobieżnie). Podpunkty z gwiazdką rozbudowują odpowiednie zadania i wymagają nieco większego nakładu pracy.

1 Adresy IP – Reaktywacja

Na trzech serwerach zbierano listy adresów IP, z których następował nietypowy ruch przychodzący. Napisz program, który je przefiltruje i wypisze:

1. adresy zarejestrowane na poszczególnych serwerach, ale bez powtórzeń – np. adres `124.194.117.151` pojawia się na serwerze nr 1 dwukrotnie, nie chcemy tego,
2. adresy powtarzające się w każdym z zestawów i ile razy,
3. adresy, które są w logach wszystkich serwerów jednocześnie,
4. całą pulę zebranych adresów – czyli te, które pojawiły się gdziekolwiek,
5. adresy, które pojawiły się wyłącznie na jednym z serwerów i nigdzie indziej (wypisz to dla każdego z trzech serwerów).

```
ip_serwer1 = ["124.194.117.151",
              "102.26.179.4",
              "124.194.117.151",
              "228.50.34.103",
              "72.64.203.50",
              "156.71.102.245",
              "67.33.135.75",
              "240.58.59.186",
              "75.245.64.35",
              "42.107.64.123"]
ip_serwer2 = ["87.60.211.200",
              "124.194.117.151",
              "39.50.47.1",
              "35.104.29.70",
              "75.245.64.35",
              "233.152.75.71",
              "196.40.247.11",
              "110.84.193.208",
              "60.69.233.205",
              "113.164.109.228"]
ip_serwer3 = ["69.24.218.219",
              "30.186.239.61",
              "215.131.155.192",
              "234.125.132.67",
              "75.245.64.35",
              "75.245.64.35",
              "167.13.89.15",
              "9.238.14.91",
              "127.97.147.85",
              "67.33.135.75",
              "129.21.227.118"]
```

Uwagi:

1. Zadanie stanie się **bardzo** proste jeśli posłużymy się zbiorami (`sets`), przydać się może także metoda `.count(...)` dla list.
2. Tak naprawdę adresy wygenerowałem losowo, więc proszę ich nie traktować jako IP faktycznych złoczyńców. 🙂

2 Szyfr „książkowy”

Szyfr książkowy (`https://pl.wikipedia.org/wiki/Szyfr_Ottendorfa`) jest to szyfr, w którym wiadomość zaszyfrowana jest w postaci zestawu liczb oznaczających lokalizację jej kolejnych liter bądź słów (zależnie od wariantu) w określonym tekście. Jedynie jednocześnie wiedząc, o który tekst chodzi i posiadając szyfr, odbiorca może odkodować przekazywaną informację.

Napisz funkcję, która będzie implementowała pewien rodzaj tego szyfru (i używający jej program). Niech przyjmuje jako argumenty nazwy dwóch plików – z tekstem i z szyfrem, po czym niech wypisuje na ekran rozszyfrowaną wiadomość. Niech plik z szyfrem w każdej linii zawiera instrukcję do odszyfrowania kolejnej litery wiadomości: trzy liczby, odpowiadające kolejno linii, wyrazowi i literze.

*A teraz w drugą stronę

Napisz drugą funkcję, która znów pobierać będzie jako argumenty nazwy plików – z tekstem oraz plik, do którego zapisany zostanie zaszyfrowana wiadomość, a także string z wiadomością do zaszyfrowania. Niech wykonuje ona operację przeciwną niż poprzednia – wyszukuje w pliku z tekstem wystąpienia kolejnych liter szyfrowanej wiadomości i wpisuje ich położenia do pliku z szyfrem. W przypadku niepowodzenia niech wypisuje, których znaków nie udało się znaleźć w tekście.

3 Archiwa `.tar`

Na jednych z poprzednich zajęć przesłałem Państwu materiały w formacie archiwum `tar`. Jest to dość stara metoda przechowywania wielu plików w jednej paczce – na tyle stara, że jej nazwa to akronim ***T**ape **AR**chive*, czyli *Archiwum Taśmowe*! To dość prosty format – polega on na sklejeniu archiwizowanych plików ze sobą. Każdy plik poprzedzony jest nagłówkiem oraz dopełniony do najbliższej wielokrotności 512 bajtów zerami. Dokładniejsze informacje są powszechnie dostępne w sieci, np.:

- `https://en.wikipedia.org/wiki/Tar_%28computing%29`
- `https://wiki.osdev.org/USTAR`

W szczególności interesują nas w nagłówku bajty:

- 0-99: nazwa pliku, dopełniona znakami ASCII `NUL` (`'\x00'`)
- 124-134: długość pliku, niestety ze względów historycznych w postaci liczby w formacie ósemkowym, której cyfry zapisane są jako ciąg ASCII

Po 512 bajtach nagłówka zaczyna się właściwa treść pliku, która jest następnie – jak wspomniano – dopełniana zerami do najbliższej wielokrotności 512 bajtów, po czym ewentualnie zaczyna się nagłówek kolejnego pliku.

Celem niniejszego zadania jest napisanie programu w Pythonie, który będzie w stanie wyodrębniać z podanego archiwum `.tar`. Dla uproszczenia ignorujemy pozostałe informacje zawarte w nagłówku (np. uprawnienia dostępu do pliku i jego właściciel) oraz zakładamy, że archiwum zawiera tylko i wyłącznie zwyczajne pliki (nie ma katalogów ani linków).

Wskazówki:

1. Chcemy operować na plikach w trybie binarnym, więc musimy dodać `'b'` do trybu otwierania.
2. Do zamiany danych binarnych na string może służyć metoda `.decode()`, zaś do usuwania niepotrzebnych znaków na końcu stringu – metoda `.rstrip(znak)`.
3. Do zamiany danych binarnych na liczbę całkowitą w formacie ósemkowym może służyć funkcja `int(dane_binarne,8)`.

*Sprawdzanie czy wyodrębniane pliki istnieją

Ulepsz napisany program w taki sposób, by przed wyodrębnieniem pliku sprawdzał czy dany plik już nie istnieje. Jeżeli istnieje, niech spyta użytkownika czy go nadpisać. W kolejnym kroku dodaj możliwość wywołania programu z argumentem (np. `--overwrite`), który spowoduje automatyczne nadpisywanie, bez pytania. Możesz także dodać drugi (np. `--keep-old-files`), który zadziała przeciwnie – wyodrębni tylko te pliki, które nie istnieją.

Wskazówki:

1. Tryb otwierania pliku w Pythonie `'w'` automatycznie nadpisuje plik, podczas gdy `'x'` zwraca błąd (`FileExistsError`) gdy plik istnieje.
2. Błąd ten można oczywiście przechwycić za pomocą konstrukcji `try`–`except`.