Programowanie I R

Zadania – seria 3.

Kolekcje: zbiór i słownik.

Zadanie 1. `lotto` – Losowanie liczb.

Napisz program `lotto`, który losuje sześć różnych liczb całkowitych z przedziału od 1 do 49 włącznie, a następnie wypisuje na standardowe wyjście wylosowane liczby w kolejności rosnącej, oddzielone spacjami.

Wskazówka 1. Wykorzystaj zbiór – dzięki temu, że każdy jego element może w nim wystąpić tylko raz, łatwiej będzie spełnić warunek nakazujący, by wylosowane liczby były różne.

Wskazówka 2. W celu wylosowania liczby całkowitej z zadanego przedziału możesz posłużyć się funkcją `randint` z modułu `random`.

Przykładowe wykonanie
Wywołanie:
Windows: `py lotto.py`
macOS / Linux: `python3 lotto.py`
Wyjście:
```
1 7 14 16 36 46
```

Zadanie 2. `words` – Liczenie wyrazów.

Napisz program `words`, który wczytuje ze standardowego wejścia oddzielone spacjami wyrazy, a następnie wypisuje te wyrazy na standardowe wyjście w kolejności alfabetycznej, każdy tylko raz, podając przy każdym z nich ilość jego wystąpień.

Przykładowe wykonanie
Wywołanie:
Windows: `py words.py`
macOS / Linux: `python3 words.py`
Wejście
```
pies kot żółw kot kot pies żółw
```
Wyjście
```
kot 3
pies 2
żółw 2
```

Zadanie 3. `coviddata` – Krzywa zachorowań na COVID–19.

Napisz program `coviddata`, wykreślający krzywą zachorowań na COVID–19 dla wybranej przez użytkownika lokalizacji. Program powinien pobierać z internetu plik z danymi dotyczącymi ilości zachorowań w formacie CSV, udostępniony pod adresem

`https://covid.ourworldindata.org/data/jhu/full_data.csv`

a następnie wypisywać listę lokalizacji – krajów, kontynentów i całego świata – dla których dane są dostępne, prosić użytkownika o wskazanie jednej z nich i rysować krzywą przedstawiającą dzienną ilość nowych zachorowań na COVID–19 w tej lokalizacji.

Szkic rozwiązania.

- Napisz kod pobierający plik z danymi i zapisujący go w bieżącym katalogu. Możesz użyć do tego celu na przykład funkcji `get` z modułu `requests` (prościej, wymagana wcześniejsza instalacja pakietu `requests`) lub funkcji `urlopen` z modułu `urllib.request` (trudniej, jednak nie wymaga instalacji).

- Wczytaj plik z danymi, używając polecenia

  ```
  lines = open("full_data.csv").readlines()
  ```

  Jakiego typu jest obiekt `lines`? Możesz to sprawdzić korzystając z funkcji `type()`.

- Zapisz w zmiennej `data` trzeci element listy `lines`. Jakiego typu jest obiekt `data`?

- Podziel łańcuch `data` na fragmenty z przecinkiem jako separatorem, wywołując metodę `split()` z odpowiednimi argumentami. Zbadaj, jak wygląda wynik działania tej funkcji i wydobądź z niego informacje o dacie, kraju oraz ilości zachorowań. Wypisz te dane, by sprawdzić, czy kod działa poprawnie.

- Zmodyfikuj swój kod tak, by zamiast analizować pojedynczą linię pliku z danymi, badał wszystkie.

  Można w tym celu po prostu analizować kolejne elementy listy `lines` (np. za pomocą pętli `for`). Rozwiązanie takie byłoby poprawne, miałoby jednak istotną wadę. Polecenie `open("full_data.csv").readlines()` wczytuje do pamięci od razu cały plik `full_data.csv` i tworzy ogromną listę `lines` zawierającą wszystkie jego linie. Proces ten w przypadku dużej ilości danych jest długotrwały i zużywa sporo pamięci operacyjnej; co więcej, gdyby plik z danymi był zbyt duży, pamięci mogłoby zabraknąć – zostałby wówczas zgłoszony wyjątek `MemoryError`, a program nie wykonałby swojego zadania.

  Bezpieczniej jest wczytywać plik z danymi linia po linii, program będzie wówczas działał poprawnie dla dowolnie dużej ilości danych. Można w tym celu posłużyć się na przykład następującym kodem (zamiast tworzyć listę `lines`):

  ```
  with open("full_data.csv") as datafile:
      for line in datafile:
          # Łańcuch line zawiera treść pojedynczej linii pliku.
  ```

  Zastosowanie polecenia `with` daje nam pewność, że po zakończeniu pracy z plikiem wszystkie związane z nim zasoby zostaną zwolnione (wcześniej nie przejmowaliśmy się tym problemem, co mogło skutkować pojawieniem się tzw. wycieku pamięci).

  Utwórz pusty słownik o nazwie `cases_pol`, a następnie przeanalizuj kolejne linie pliku z danymi, sprawdzając, czy dana linia dotyczy Polski, i – jeśli tak jest – dodaj do słownika `cases_pol` element, którego kluczem będzie data (np. w postaci łańcucha znaków), zaś wartością – liczba zachorowań. Narysuj wykres danych zawartych w słowniku `cases_pol`, by sprawdzić, czy Twój kod działa poprawnie.

  Postaraj się tak skonfigurować wykres, by był on czytelny i estetyczny. W celu zapewnienia poprawnego wyświetlania etykiet na osi odciętych konieczne może się okazać przechowywanie dat w słowniku `cases_pol` nie w postaci łańcuchów tekstowych, lecz za pomocą obiektów dedykowanego datom typu, np. typu `datetime` zdefiniowanego w module `datetime`.

- Zmodyfikuj program tak, by analizował dane o zachorowaniach dla wybranej przez użytkownika lokalizacji.

  Najprostszym sposobem realizacji tego celu wydaje się być zastąpienie słownika `cases_pol` słownikiem `cases` mającym strukturę „słownika słowników”: każdy jego klucz powinien być nazwą jednej z lokalizacji, dla których dane o zachorowaniach są dostępne, zaś wartość odpowiadająca danemu kluczowi

– słownikiem zawierającym dane o zachorowaniach w lokalizacji, na którą wskazuje ten klucz, zbudowanym analogicznie, jak omówiony szczegółowo w poprzednim punkcie słownik `cases_pol`. Obiekt `cases["Italy"]` będzie wówczas słownikiem zawierającym dane o zachorowaniach we Włoszech, obiekt `cases["Poland"]` – słownikiem z danymi o zachorowaniach w Polsce (dokładnie tym samym, którym wcześniej był `cases_pol`) etc.

Rozwiązanie to, choć poprawne, nie jest optymalne, w pamięci operacyjnej powstanie bowiem ogromna struktura danych, przechowująca wszystkie informacje zawarte w pliku z danymi. Gdy informacji tych będzie dużo, mogą wystąpić te same problemy, o których wspominaliśmy w poprzednim punkcie.

Lepszym rozwiązaniem będzie przeprowadzenie analizy danych dwuetapowo. Na początku możemy wydobyć z pliku z danymi wyłącznie informację o tym, dla jakich lokalizacji dostępne są dane, następnie poprosić użytkownika o wskazanie interesującej go lokalizacji i ponownie przejrzeć plik, odczytując informacje o zachorowaniach wyłącznie dla tej lokalizacji.

Kierując się tą ideą, zmodyfikuj kod programu w następujący sposób. Zamiast słownika `cases_pol` program powinien tworzyć pusty zbiór o nazwie `locations`, a potem analizować kolejne linie pliku z danymi, wydobywając z każdej z nich informację o tym, jakiej lokalizacji dotyczą zawarte w niej dane, i dodawać nazwę tej lokalizacji do zbioru `locations` (zastosowanie zbioru będzie tu sporym ułatwieniem – zbiór, w przeciwieństwie do np. listy, utożsamia ze sobą identyczne elementy, nie pozwalając na pojawienie się duplikatów). Program powinien następnie wypisać elementy zbioru `locations` i poprosić użytkownika o wybranie jednego z nich. Gdy użytkownik poda poprawną nazwę lokalizacji (np. `Poland`), program powinien utworzyć słownik o nazwie `cases` zawierający dane na temat zachorowań w tej lokalizacji (skonstruowany analogicznie, jak omówiony szczegółowo w poprzednim punkcie słownik `cases_pol`) i – na jego podstawie – narysować krzywą zachorowań dotyczącą tej lokalizacji.

Zadanie 4. Ciąg Collatza.

Niech k będzie liczbą naturalną. *Ciągiem Collatza* nazywamy ciąg $\left(c_n^k\right)_{n=0}^{\infty}$ określony wzorem

$$c_n^k \stackrel{\text{def}}{=} \begin{cases} k, & \text{gdy } n = 0, \\ \frac{1}{2}c_{n-1}, & \text{gdy } c_{n-1} \text{ jest liczbą parzystą}, \\ 3c_{n-1}+1, & \text{gdy } c_{n-1} \text{ jest liczbą nieparzystą}. \end{cases}$$

Na przykład

$$\begin{aligned} \left(c_n^{12}\right) &= (12,\, 6,\, 3,\, 10,\, 5,\, 16,\, 8,\, 4,\, 2,\, \mathbf{1},\, \textit{4},\, \textit{2},\, \textit{1},\, \textit{4},\, \textit{2},\, \textit{1},\, \ldots)\,, \\ \left(c_n^{13}\right) &= (13,\, 40,\, 20,\, 10,\, 5,\, 16,\, 8,\, 4,\, 2,\, \mathbf{1},\, \textit{4},\, \textit{2},\, \textit{1},\, \textit{4},\, \textit{2},\, \textit{1},\, \ldots)\,, \\ \left(c_n^{15}\right) &= (15,\, 46,\, 23,\, 70,\, 35,\, 106,\, 53,\, 160,\, 80,\, 40,\, 20,\, 10,\, 5,\, 16,\, 8,\, 4,\, 2,\, \mathbf{1},\, \textit{4},\, \textit{2},\, \textit{1},\, \textit{4},\, \textit{2},\, \textit{1},\, \ldots)\,. \end{aligned}$$

Gdy pewien wyraz ciągu Collatza ma wartość 1, wyrazy następujące po nim mają wartości 4, 2, 1, 4, 2, 1, ... – sekwencja 4, 2, 1 powtarza się w nieskończoność. Przypuszcza się, że ciąg Collatza $\left(c_n^k\right)$ osiąga wartość 1 dla dowolnego k, hipoteza ta pozostaje jednak problemem otwartym.

a) Napisz program `collatz`, który prosi użytkownika o podanie liczby naturalnej k, a następnie oblicza i wypisuje początkowe wyrazy ciągu Collatza $\left(c_n^k\right)$ aż do pierwszego z wyrazów o wartości 1 włącznie.

b) Napisz program `lcollatz` znajdujący tę liczbę naturalną $k < 10^6$, dla której ciąg Collatza $\left(c_n^k\right)$ ma najwięcej wyrazów występujących przed pierwszym z wyrazów o wartości 1.

Opracowanie: Bartłomiej Zglinicki.