Lista zadań nr 4

Pliki tekstowe, wyjątki, serializacja.

Zadanie 1 Napisz program symulujący **magiczną kulę** - zabawkę, która wyświetla losowe odpowiedzi na zadawane pytania. W załączonym pliku *magiczna_kula.txt* znajduje się 12 odpowiedzi, takich jak: `"Nie sądzę."`, `"Oczywiście, że tak!"`, `"Nie wiem."`.

Program powinien:

- odczytać zawartość pliku i zapisać wszystkie odpowiedzi do listy,
- poprosić użytkownika o wpisanie pytania,
- wyświetlić losowo wybraną odpowiedź z listy,
- powtarzać powyższe kroki tak długo, jak użytkownik chce kontynuować (np. do momentu wpisania komendy `"koniec"`).

Zadbaj o czytelność kodu i obsłuż sytuację, w której plik z odpowiedziami nie istnieje.

Zadanie 2 Napisz program, który będzie symulował prosty kalkulator statystyczny. Program powinien działać w dwóch wersjach:

Wersja podstawowa (interaktywna):

- Poproś użytkownika w pętli `while` o wprowadzenieliczby rzeczywistej.
- Użytkownik kończy wprowadzanie, naciskając klawisz `Enter` bez podawania liczby.
- Program powinien zebrać wszystkie liczby do listy, a następnie wypisać:
  - wprowadzone liczby,
  - ilość liczb,
  - sumę,
  - najmniejszą i największą wartość (użyj funkcji `min()` i `max()`),
  - średnią arytmetyczną,
  - medianę (można użyć modułu `statistics`).

Przykładowa sesja programu:

```
podaj liczbę lub naciśnij Enter aby zakończyć: 5
podaj liczbę lub naciśnij Enter aby zakończyć: 3
podaj liczbę lub naciśnij Enter aby zakończyć: 3
podaj liczbę lub naciśnij Enter aby zakończyć: 2
podaj liczbę lub naciśnij Enter aby zakończyć: 4
podaj liczbę lub naciśnij Enter aby zakończyć: 6
podaj liczbę lub naciśnij Enter aby zakończyć:
liczby: [5, 3, 3, 2, 4, 6]
ilość = 6   suma = 23   najmniejsza = 2   największa = 6
średnia = 3.8333333333333335   mediana = 3.5
```

Wersja alternatywna (plikowa):

Program powinien wczytywać dane z pliku *numbers.txt*, w którym znajduje się 100 wierszy. Każdy wiersz zawiera ciąg liczb rzeczywistych oddzielonych spacją.

Dla każdego wiersza program powinien obliczyć i wyświetlić:

- ilość liczb,
- sumę,
- minimum i maksimum,
- średnią,
- medianę.

Zadbaj o poprawne otwieranie pliku oraz obsługę ewentualnych błędów.

Zadanie 3 Opracuj i przetestuj program w języku Python, który umożliwia konwersję wiadomości tekstowej zapisanej wielkimi literami alfabetu łacińskiego na kod Morse'a oraz odwrotnie - z kodu Morse'a na tekst.

Wymagania funkcjonalne:

- Program powinien pytać użytkownika o:
  - rodzaj tłumaczenia: z tekstu na kod Morse'a lub z kodu Morse'a na tekst,
  - nazwę pliku wejściowego zawierającego dane do przetworzenia,
  - nazwę pliku wynikowego.
- W przypadku tłumaczenia z tekstu na kod Morse'a:
  - każdy zakodowany znak powinien zostać zapisany w osobnym wierszu,

- jedna pusta linia oznacza przerwę między wyrazami,
- dwie puste linie oznaczają koniec zdania.

- W przypadku tłumaczenia z kodu Morse'a na tekst:
  - znaki wczytywane są wiersz po wierszu,
  - pojedyncza pusta linia rozdziela wyrazy,
  - dwie puste linie rozdzielają zdania.
- Program powinien wykorzystywać słowniki do reprezentacji kodowania.
- Obsłuż sytuację, w której plik wejściowy nie istnieje.

Program powinien działać wyłącznie na literach alfabetu łacińskiego (bez polskich znaków i cyfr).

Tabela 1: Reprezentacja liter alfabetu łacińskiego w kodzie Morse'a.

A	• –	N	– •
B	– • • •	O	– – –
C	– • – •	P	• – – •
D	– • •	Q	– – • –
E	•	R	• – •
F	• • – •	S	• • •
G	– – •	T	–
H	• • • •	U	• • –
I	• •	V	• • • –
J	• – – –	W	• – –
K	– • –	X	– • • –
L	• – • •	Y	– • – –
M	– –	Z	– – • •

Zadanie 4 Plik *dl.txt* zawiera archiwalne wyniki losowań Dużego Lotka od roku 1957. Każdy wiersz pliku zawiera 6 liczb całkowitych oznaczających wylosowane liczby w danym losowaniu.

Napisz program w języku Python, który:

- poprosi użytkownika o podanie swoich 6 unikalnych liczb z zakresu od 1 do 49 (waliduj poprawność danych),

- wczyta wyniki losowań z pliku *dl.txt*,
- porówna podane liczby z każdym historycznym losowaniem i zliczy:
  - liczbę trafionych **szóstek**,
  - liczbę **piątek**,
  - liczbę **czwórek**,
  - liczbę **trójek**.
- wyświetli raport końcowy z wynikami zliczeń.

Zadbaj o czytelność kodu, poprawne wczytywanie pliku oraz uwzględnij możliwość jego braku. W razie potrzeby poinformuj użytkownika o błędach wejścia.

Zadanie 5 Plik *bond.txt* zawiera informacje o filmach z serii o Jamesie Bondzie: tytuł filmu, imię i nazwisko aktora odgrywającego rolę agenta 007, datę premiery w Wielkiej Brytanii oraz datę premiery w USA. Daty zapisane są w formacie `dd/mm/rrrr`.

Napisz program, który:

- wczyta dane z pliku *bond.txt*,
- przekształci obie daty do formatu `rrrr-mm-dd`,
- zapisze przetworzone dane do nowego pliku *bond2.txt*, zachowując strukturę oryginalnego pliku.

Użyj funkcji z modułu `datetime` do przetwarzania dat.

Zadanie 6 Napisz skrypt, który wczyta dane z pliku *bond.txt* i wyświetli je w następującej uproszczonej postaci:

```
1962 Dr. No
1963 Pozdrowienia z Rosji
1964 Goldfinger
-- kolejne wiersze --
2015 Spectre
```

Dla każdego filmu należy wyświetlić rok brytyjskiej premiery oraz tytuł filmu. Dane zapisz również do pliku *bond_info.txt*.

Zadanie 7 Przetwórz plik *bond.txt* tak, aby dla każdego filmu została wyświetlona liczba dni, jakie upłynęły pomiędzy datą premiery w Wielkiej Brytanii a datą premiery w USA.

Wynik powinien mieć postać:

```
Dr. No 215
Pozdrowienia z Rosji 181
-- kolejne wiersze --
```

Wskazówka: użyj klasy `datetime.date` do obliczenia różnicy dni pomiędzy dwiema datami.

Zadanie 8 Napisz program **Quiz**, który sprawdza wiedzę użytkownika za pomocą serii pytań testowych.

Pytania powinny być przechowywane w pliku tekstowym o następującej strukturze: każdy blok składa się z dokładnie 8 wierszy i odpowiada jednemu pytaniu. Plik może zawierać dowolną liczbę takich bloków.

Struktura jednego bloku:

```
<kategoria>
<pytanie>
<odpowiedź 1>
<odpowiedź 2>
<odpowiedź 3>
<odpowiedź 4>
<numer prawidłowej odpowiedzi (1-4)>
<wyjaśnienie>
```

Program powinien:

- losować kolejność pytań z pliku (można użyć modułu `random`),
- wyświetlać pytanie oraz cztery możliwe odpowiedzi,
- pobierać odpowiedź użytkownika i sprawdzać jej poprawność,
- po każdej odpowiedzi wyświetlać informację zwrotną wraz z wyjaśnieniem,
- zliczać i wyświetlić końcową liczbę punktów zdobytych przez użytkownika.

Na potrzeby zadania utwórz przykładowy plik z przynajmniej 10 pytaniami dotyczącymi języka Python.

Zadbaj o poprawną walidację danych wejściowych oraz czytelność interfejsu tekstowego.

Zadanie 9 Zaprojektuj program w języku Python, który umożliwia użytkownikowi rozegranie gry **„papier, kamień, nożyce”** przeciwko komputerowi.

Opis działania programu:

1. Po uruchomieniu program generuje losową liczbę z przedziału 1–3, gdzie:
   - 1 oznacza `kamień`,
   - 2 oznacza `papier`,
   - 3 oznacza `nożyce`.

   Wynik komputera nie jest od razu wyświetlany.
2. Użytkownik wybiera jedną z opcji: `kamień`, `papier` lub `nożyce`, wpisując ją z klawiatury.
3. Program wyświetla, co wybrał komputer.
4. Następnie wyświetlany jest wynik rozgrywki zgodnie z poniższymi regułami:
   - `kamień` wygrywa z `nożycami` (kamień tępi nożyce),
   - `nożyce` wygrywają z `papierem` (nożyce tną papier),
   - `papier` wygrywa z `kamieniem` (papier owija kamień),
   - w przypadku remisu rozgrywka jest powtarzana do momentu wyłonienia zwycięzcy.
5. Program umożliwia wielokrotną rozgrywkę - użytkownik może grać wiele razy.

Dodatkowe wymagania:

- Każda rozgrywka powinna być zapisywana w pliku *games.txt*, gdzie każdy wiersz zawiera wybór użytkownika oraz wybór komputera, np.:

```
papier nożyce
kamień kamień
kamień nożyce
kamień papier
-- kolejne wiersze --
```

- Program przed rozpoczęciem gry oraz po zakończeniu serii rozgrywek powinien analizować plik *games.txt* i podawać dotychczasowy bilans:

- procent wygranych przez użytkownika,
- procent wygranych przez komputer,
- procent remisów.

Zadbaj o czytelność kodu oraz walidację poprawności danych wejściowych od użytkownika.

Zadanie 10 Napisz moduł Pythona zawierający dwie funkcje:

- `unique_words(filename)` - przyjmuje nazwę pliku tekstowego i zwraca strukturę danych (np. zbiór) zawierającą wszystkie unikalne wyrazy występujące w pliku;
- `words_counter(filename)` - przyjmuje nazwę pliku tekstowego i zwraca strukturę danych (np. słownik), w której kluczami są wyrazy, a wartościami - liczby ich wystąpień w pliku.

Następnie napisz dwa osobne skrypty korzystające z powyższego modułu:

Skrypt 1 - analiza zasobu słów w dziełach Szekspira (*hamlet.txt*, *makbet.txt*, *otello.txt*):

- dla każdego pliku wygeneruj strukturę danych z unikalnymi wyrazami,
- wypisz liczbę różnych słów w każdym z plików,
- wypisz liczbę słów wspólnych dla każdej z par plików: *hamlet.txt–makbet.txt*, *hamlet.txt–otello.txt*, *makbet.txt–otello.txt*.

Skrypt 2 - najczęstsze wyrazy:

- zdefiniuj funkcję `most_common(counter, n)`, która przyjmuje słownik (taki jak wynik działania `words_counter()`) oraz liczbę całkowitą `n`, i zwraca listę n najczęściej występujących wyrazów w postaci listy krotek:
  `[(wyraz_1, liczba_1), (wyraz_2, liczba_2), ...,`
  `(wyraz_n, liczba_n)]`
- użyj tej funkcji do wypisania 10 najczęstszych wyrazów dla każdego z plików: *hamlet.txt*, *makbet.txt*, *otello.txt*,
- wyświetl dane w formie tabeli: 3 kolumny (po jednej dla każdego dzieła), 10 wierszy - jak w poniższym przykładzie:

```
t = 'wyraz1', 123, 'wyraz2', 563, 'wyraz3', 9876
print('|{0:<10}:{1:4} |{2:<10}:{3:5} |{4:<10}:{5:5} |'.format(*t))
```

Zadbaj o:

- normalizację tekstu (np. zamianę na małe litery, usuwanie znaków interpunkcyjnych),
- czytelność i modularność kodu,
- poprawne otwieranie plików oraz obsługę błędów,
- opcjonalnie: możliwość zapisania wyników do pliku *wyniki.txt*.

KOLEJNE, NOWE ZADANIA ZOSTANĄ OPUBLIKOWANE WKRÓTCE!!