Zadanie 2.1 Sklep

Uzupełnij tworzony w czasie zajęć program sklep o następujące szczegóły:

1. Napisz program w formie programu tekstowego z menu – podobnie, jak w przykładzie „program_geometryczny”. Program ma w pętli pytać „Co chcesz zrobić?”, a do wyboru mają być takie opcje jak
   - Q – zakończ
   - Z – zakupy
   - C – ustaw cenę produktu (zmienia wartość w cenniku; może też służyć do dodawania nowych produktów)
   - U – usuń towar z cennika (operacja del na słowniku)

2. Dla każdego towaru pamiętaj nie tylko jego cenę, ale także stan magazynowy (ile jest dostępnych sztuk). Każdy zakup powinien zmniejszać stan, a próba zakupu większej liczby sztuk, niż dostępna, powinna skończyć się niepowodzeniem (program ma wypisać np. "Brak wystarczającej liczby towarów").
   Zdefiniuj operację
   - D - dostawa towaru, w której program pyta o nazwę towaru oraz liczbę sztuk i zwiększa stan towaru o podaną liczbę sztuk.

   Ma to działać tylko dla towarów, które są zdefiniowane w cenniku.
3. Informacja o stanie magazynowym ma być zapisywana do pliku i odczytywana z pliku csv wraz z nazwami i cenami towarów.
   Zdefiniuj operacje
   - R – odczyt danych z pliku
   - W – zapis aktualnego cennika i ilości produktów do pliku.

Dla chętnych:

4. Zezwól na ceny z ułamkiem (z groszami), przy czym lepsze, niż używanie typu float (który może dawać niepoprawne wyniki w obliczeniach), jest użycie typu Decimal.

Zadanie 2.2 ***Logi*** *(zadanie z podręcznika)*

Napisz program wczytujący plik z logami aktywności użytkowników w systemie: **logs.txt**. Są tam zapisane momenty zalogowania i wylogowania, zakładając, że czas jest mierzony w sekundach od momentu uruchomienia systemu i w kolejności następowania zdarzeń są one zapisane w pliku. Można przyjąć, że dane są sensowne, m.in. po każdym zalogowaniu nastąpi kiedyś moment wylogowania.

Na podstawie wczytanych danych wyświetl informację o sumarycznym czasie przebywania każdego użytkownika w systemie.

Czas przebywania w systemie:
Adam : 4032 s
Bartek : 3607 s
Celina : 3348 s

itd

Zadanie 2.3 ***Analiza danych "zawodnicy"***

Korzystając z pliku CSV z danymi skoczków narciarskich (zawodnicy.csv) napisz programy, które wczytują ten plik i:

1. Oblicza ogólne statystyki dot. zawodników z całego pliku, na które składają się:
   - średni wzrost wszystkich zawodników,
   - suma wag wszystkich zawodników,
   - informacja, kto jest najwyższy, a kto najniższy i jaki jest ich wzrost,
   - informacja, kto jest najcięższy, a kto najlżejszy i jaka jest ich waga.
2. Pozwól użytkownikowi podać kraj (trzyliterowy skrót) i oblicz analogiczne informacje dla wskazanego kraju.
3. Dla każdego kraju występującego w pliku podaje liczbę zawodników z tego kraju:
   ```
   AUT - 2
   FIN - 3
   GER - 5
   NOR - 3
   POL - 3
   USA - 1
   ```
4. Dla każdego kraju występującego w pliku oblicza takie statystyki, jak w punkcie 1.

Zadanie 2.4 ***Trójki Pitagorejskie***

Wygeneruj trójki (tuple trzyelementowe) liczb całkowitych a b c, które tworzą trójkąt pitagorejski (czyli $a^2 + b^2 = c^2$), ale bez powtórzeń i tylko takich, że $a < b < c$, dla liczb do podanego zakresu (np. 1000). Postaraj się zrobić to za pomocą wyrażenia listotwórczego.

Istnieją tu rozwiązania bardziej i mniej wydajne ;-) – do przemyślenia.

Zadanie 2.5 Funkcje liczbowe

Napisz następujące funkcje:

1. **suma_cyfr**(n)
   Zwraca sumę wartości cyfr, z których składa się liczba n. Np. dla parametru 1023 wynikiem powinno być 6.
   n ma być typu int , a nie str.
2. **fib**(n)
   Zwraca n-tą liczbę Fibonacciego, gdzie liczby Fibonacciego są zdefiniowane następująco:
   fib(0) = 0
   fib(1) = 1
   fin(n) = fib(n-2) + fib(n-1)
   Co oznacza, że początkowe liczby Fibonacciego to: 0 1 1 2 3 5 8 13 21 34
   Postaraj się do następnego spotkania ;) obliczyć tysięczną liczbę Fibonacciego
3. **najmniejszy_dzielnik**(liczba)
   zwraca najmniejszy dzielnik tej liczby większy niż 1.
   Przykładowo: najmniejszy_dzielnik(15) == 3, najmniejszy_dzielnik(17) == 17
4. **czy_pierwsza**(liczba)
   Sprawdza czy liczba jest pierwsza i zwraca **True** albo **False**.

   Liczba pierwsza to taka liczba naturalna, która ma dokładnie dwa różne dzielniki naturalne (1 i samą siebie). Np. 13 jest liczbą pierwszą, a 15 nie jest, gdyż dzieli się także przez 3 i 5.
   0 i 1 nie są pierwsze. Przykład dużej liczby pierwszej do sprawdzenia: 2147483647 (2**31 - 1).

Zadanie 2.6 Funkcja statystyki

Napisz funkcję **statystyki**(lista), która:

- jako parametr otrzymuje listę liczb (lub inne „iterowalne” źródło danych)
- a jako wynik zwraca krotkę (tuple) pięciu liczb: liczba elementów, suma wartości, średnia arytmetyczna, minimalna wartość, maksymalna wartość.

Zachęcam do samodzielnego obliczenia statystyk za pomocą pojedynczej pętli i odpowiednich zmiennych, a nie wykorzystywania gotowych funkcji Pythona.

Przetestuj działanie funkcji na kilku przykładowych listach. Nie musisz pisać programu, który czyta dane od użytkownika! Sednem zadania jest napisanie samej funkcji.

Zadanie 2.7 Klasy dla ogłoszeń

Zaprojektuj klasy odpowiadające ogłoszeniom (w zmyślonym serwisie ogłoszeniowym).

Klasa `Ogloszenie`: każde ogłoszenie zawiera takie informacje jak: tytuł ogłoszenia, opis, cena, data wystawienia, dane wystawiającego (imię lub nazwa, telefon, email).

Stwórz podklasy odpowiadające ogłoszeniom różnych typów, na przykład:

`OgloszenieSamochodowe` - dziedziczy z `Ogloszenie` i dodaje takie informacje jak marka i model samochodu, rocznik, przebieg, rodzaj paliwa, moc silnika itp

`OgloszenieMieszkaniowe` - dziedziczy z `Ogloszenie` i dodaje takie informacje jak rodzaj nieruchomości (możesz też pomyśleć o kolejnych podklasach...), metraż, liczba pokoi, czy jest winda, piętro itp.

W klasach zdefiniuj odpowiednio metody `__init__` i `__str__`. Możesz dodać inne metody wedle uznania. Napisz przykładowy program, który tworzy obiekty i je wypisuje.

Zadanie 2.8 ElectricCar - zadanie z podręcznika

Zaimplementuj klasę ElectricCar odwzorowującą zachowanie samochodu elektrycznego. Klasa powinna umożliwiać pokonanie zadanego dystansu, który nie może przekroczyć maksymalnego zasięgu zdefiniowanego dla samochodu. Samochód powinien mieć także możliwość naładowania baterii do pełna.

```
>>> car = ElectricCar(100)
>>> car.drive(70)
70
>>> car.drive(50)
30
>>> car.drive(50)
0
>>> car.charge()
>>> car.drive(50)
50
```

Dodatkowe wyjaśnienia:

- 100 na początku oznacza maksymalny zasięg w km, można podać inny niż 100. To właśnie do tej wartości ładowana jest bateria, gdy wywołamy charge().
- drive(dystans) oznacza, że „próbujemy” przejechać podany dystans, a wynikiem (return) metody jest informacja, ile udało się faktycznie przejechać. Jeśli baterii wystarcza na mniej, to wtedy przejedziemy mniej (jak w przykładzie z wynikiem 30). Gdy bateria jest już pusta, wynikiem jest 0.

Zadanie 2.9 Klasa dla ułamków

(do zrobienia raczej po ost. zajęciach, bo muszę jeszcze pokazać Wam „magiczne metody”)

Stwórz klasę, której obiekty reprezentują liczby wymierne w postaci ułamkowej. Klasa powinna implementować **dokładne operacje na ułamkach** (nie poprzez konwersję na float). Podobnie jak robiliśmy z Vectorem, utwórz klasę, która zwraca nowe obiekty jako wyniki swoich operacji.

Nie dopuszczaj do sytuacji, aby w mianowniku znalazło się zero – wyrzucaj wyjątek, a także utrzymuj mianownik dodatni (gdy ktoś próbuje podać ujemny, to wpisz dodatni i zamień znak licznika). Operacje arytmetyczne powinny zwracać wynik w postaci maksymalnie skróconej.

Proponowane elementy klasy:

- tworzenie: `Fraction(nominator=0, denominator=1)`
- atrybuty `nominator`, `denominator` dają wartość licznika i mianownika
- `__str__ , __repr__`
- operacje matematyczne **+ - * /** za pomocą magicznych metod __add__, __sub__ itd.
- porównania == != < <= > >= działające w oparciu o wartości, ale bez wyliczania floatów,
- `__float__` i `__int__` – konwersja na typ float (z maksymalną precyzją) i int (część całkowita)
- `__bool__` – zwraca True jeśli licznik nie jest zerem

Metody zwracające wyniki – można je oznaczyć jako `@property`

- `shortened()` – zwraca ułamek o tej samej wartości, skrócony w miarę możliwości (o największy wspólny dzielnik licznika i mianownika)
- `reciprocal()` – zwraca odwrotność ułamka (nowy obiekt)

 # Wraz z dodawaniem kolejnych metod, testuj je za pomocą pytest.

Zadanie 2.10 Metoda d'Honta

Zaimplementuj funkcję w Pythonie, która realizuje liczbowy podział mandatów między komitety wyborcze zgodnie z „regułą d'Honta”.

Funkcja jako parametry otrzymuje: słownik, w którym dla nazwy komitetu podany jest wynik głosowania, oraz liczbę mandatów do podziału. Jako wynik funkcja powinna zwrócić słownik, gdzie dla nazwy komitetu podana będzie liczba zdobytych mandatów.

Przykład:

wyniki = {"czerwoni": 12000, "zieloni": 7000, "niebiescy": 5000, "pomarańczowi": 3000}

mandaty = dhont(wyniki, 5)

assert mandaty == {"czerwoni": 3, "zieloni: 1, "niebiescy": 1, "pomarańczowi": 0}

Jak działa metoda d'Honta

Wyniki poszczególnych "partii" są dzielone przez kolejne liczby naturalne (1, 2, 3, ...) i z uzyskanych ilorazów branych jest tyle największych, ile jest mandatów do podziału.

Tabela pokazuje ilorazy dla przykładowych danych. Pogrubione jest 5 największych - mandaty idą do tych partii:

	wynik / 1	wynik / 2	wynik / 3	wynik / 4	wynik / 5
czerwoni	**12000**	**6000**	**4000**	3000	2400
zieloni	**7000**	3500	2333	1750	
niebiescy	**5000**	2500	1667		
pomarańczowi	3000	1500	1000		

W przypadku pojawienia się równego ilorazu (np. liczba 3000 gdyby tu było 7 mandatów do podziału), mandat zdobywa partia o większej bezwzględnej liczbie głosów. W przypadku równej liczby głosów możecie przyjąć, że decyduje kolejność w słowniku.

Zgodnie z polską ordynacją wyborczą do Sejmu, taki podział jest dokonywany niezależnie w każdym z 41 okręgów wyborczych pomiędzy te komitety, które w skali kraju przekroczyły próg (5% dla partii, 8% dla koalicji). Liczba mandatów do zdobycia w tych okręgach waha się od 7 (Częstochowa) do 20 (Warszawa).

Pan Tadeusz

Napisz zestaw programów, które analizują wskazany plik tekstowy, np. plik `pan_tadeusz.txt`. W większości przypadków trzeba plik podzielić na słowa, do czego można użyć takich technik jak metoda `split` albo wyrażenia regularne.

Przypomnę, że `linia.split(' ,.;!?()')` podzieli linię na słowa biorąc pod uwagę nie tylko spacje, ale także inne znaki (w P.T. jest ich jeszcze więcej).

W razie problemów - piszcie maila, pomogę ;)

Zadanie 2.11 Policz wybrane słowo

Program, który pyta użytkownika o słowo i dla podanego słowa liczy ile razy to słowo występuje w pliku. Przykładowa sesja:

```
Podaj słowo: Tadeusz
Słowo Tadeusz występuje 107 razy.
```

Zadanie 2.12 Policz wszystkie słowa

Napisz program, który czyta plik tekstowy i wylicza oraz wypisuje bez powtórzeń wszystkie słowa występujące w pliku wraz z informacją ile razy dane słowo występuje. Na przykład w ten sposób (drobny wycinek):

```
          Tace ->     1
        Tadeusz ->   107
       Tadeusza ->    54
     Tadeuszek ->     1
```

W podstawowej wersji nie przejmuj się kolejnością wypisywanych informacji.

2.12.1 * Sortowanie

Sprowadzaj wszystkie słowa do małych liter i licz jednolicie bez podziału na małe i duże litery.

Posortuj wypisywane wyniki na dwa sposoby:

a) alfabetycznie

b) według liczby wystąpień.

*Zadanie 2.13 * Odmiana słów*

Pod adresem **zil.ipipan.waw.pl/PoliMorf** znajdują się zasoby dotyczące odmiany wyrazów w języku polskim. Plik `PoliMorf-0.6.7.tab` (lub nowsza wersja) zawiera relację między słowem a jego formą podstawową. W pliku tym pierwsza kolumna zawiera słowo być może odmienione, a druga kolumna zawiera jego formę bazową.

Wczytaj dane z tego pliku do odpowiednich struktur Pythona (list / set / dict) tak, aby dało się wyszukiwać odmiany dla form podstawowych, a formy podstawowe dla odmian i aby działało to w przyzwoitym czasie (wczytywanie może chwilę potrwać, ale znalezienie odmiany, gdy dane są już wczytane, powinno być szybkie – raczej sekunda niż minuta :))

Najporządniej byłoby utworzyć klasę, której obiekt zawiera odpowiednio przygotowaną „bazę wiedzy” nt. odmiany słów. Obiekt tej klasy tworzy się na podstawie pliku takiego jak `PoliMorf-0.6.7.tab` . Obiekt powinien umożliwiać znalezienie formy podstawowej (lub kilku kandydatów) dla podanego słowa odmienionego oraz odczytanie listy słów odmienionych na podstawie formy bazowej.

Przykładowe użycie tej klasy mogłoby wyglądać tak (ale szczegóły nie muszą wyglądać identycznie):

```
>>> baza = BazaOdmiany.wczytaj("PoliMorf-0.6.7.tab")
>>> baza.znajdzFormyPodstawowe("Tadeusza")
['Tadeusz']
>>> baza.znajdzOdmiany("Tadeusz")
['Tadeusz', 'Tadeuszowi', 'Tadeusza'…...
```

Korzystając z tej klasy napisz program, który w pętli odczytuje od użytkownika kolejne słowa i dla podanego słowa wyświetla jego formę podstawową lub informację, że nie znaleziono.

Ten sam wyraz może być formą odmienioną dla kilku form bazowych, np. „dam” może pochodzić od „dać”, ale również od „dama”. Można to uwzględnić we własnym modelu (wersja zaawansowana) lub upraszczając przyjąć, że wybieracie tylko jedną formę podstawową dla danej formy odmienionej.

*Zadanie 2.14 * Pan Tadeusz z odmianą wyrazów*

Zrealizuj zadanie z liczeniem ilości poszczególnych słów w Panu Tadeuszu, ale sprowadzaj wszystkie słowa do ich formy podstawowej (albo pierwszej formy podstawowej znalezionej w pliku, jeśli jest kilka kandydatur). Przykładowo zamiast całej serii odmienionych Tadeuszów

```
Tadeusz          →  107
Tadeusza         →   54
Tadeuszem        →    2
Tadeuszowi       →    5
Tadeuszu         →    6
```

powinien być jeden wpis

```
Tadeusz → 174
```