

## 2. Po drugim tygodniu

### Zadanie 2.1 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.  
parametr n jest typu int
2. **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 ( $2^{31} - 1$ ).

3. **fib(n)**  
Zwraca n-tą liczbę Fibonacciego, gdzie liczby Fibonacciego są zdefiniowane następująco:  
 $fib(0) = 0$   
 $fib(1) = 1$   
 $fib(n) = fib(n-2) + fib(n-1)$   
Co oznacza, że początkowe liczby Fibonacciego to: 0 1 1 2 3 5 8 13 21 34  
Inaczej mówiąc, każda kolejna liczba Fibonacciego jest sumą dwóch poprzednich.  
Postaraj się do następnego spotkania ;) obliczyć tysięczną liczbę Fibonacciego

**Zadanie 2.2 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 2.3 Trójki Pitagorejskie**

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

**Zadanie 2.4 Łączenie słowników**

Napisz funkcję `merge(dict1, dict2)`, która pobiera jako parametry dwa słowniki i w wyniku zwraca słownik. Zasada działania powinna być taka:

- Jeśli jakiś klucz  $K$  występuje tylko w jednym ze słowników z wartością  $V$ , to w wyniku powinien pojawić się wpis z takim kluczem i taką samą wartością  $K: V$ .
- Jeśli jakiś klucz  $K$  występuje w jednym słowniku z wartością  $V1$ , a w drugim z wartością  $V2$ , to w wynikowym słowniku powinien znaleźć się jeden wpis z kluczem  $K$  i wartością  $V1+V2$ .

Wersja zaawansowana dla chętnych – `merge(f, *dicts)`

- Pierwszym parametrem funkcji `merge` jest „funkcja mergująca”  $f$ , która przyjmuje dwa parametry i zwraca jeden wynik
- Następnie `merge` może przyjąć dowolnie wiele słowników
- Zasada działania jest podobna, jak opisana wyżej, ale w przypadku występowania tego samego klucza w wielu słownikach ich wartości się są sumowane matematycznym  $+$ , ale jest wywoływana funkcja przekazana jako pierwszy argument.
- Przykład użycia:  

```
>>> merge(lambda x,y: x*y, {'a':10, 'b':20}, {'b':3, 'c': 4})  
{'a':10, 'b':60, 'c':4}
```

**Zadanie 2.5 Sklep**

Uzupełnij tworzony w czasie zajęć program sklep o następujące szczegóły:

1. Program powinien działać na zasadzie pętli, w której pyta użytkownika o akcję do wykonania, a użytkownik wybiera za pomocą odpowiedniej litery – podobnie, jak zrobiliśmy w przykładzie "geometria".  
Proponowane operacje na początek:
  - Q - wyjście z programu, a przy tym wypisanie sumarycznej kwoty za towary zakupione w czasie sesji
  - K - zakupy - pytanie o nazwę towaru i liczbę sztuk
  - C - zmiana ceny towaru - pytanie o nazwę towaru oraz nową cenę i aktualizacja słownika
  - N - dodanie nowego towaru do cennika
  - U - usunięcie towaru z cennika
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.
3. Informacje o towarach, ich cenach i stanie magazynowym ma być zapisywana do pliku i odczytywana z pliku tekstowego (w każdej linii kilka wartości rozdzielonych separatorem, jak w CSV).  
Zdefiniuj operacje
  - R – odczyt danych z pliku
  - W – zapis aktualnego cennika i ilości produktów do pliku

Proponuję takie rozwiązanie, że użytkownik może podać własną nazwę pliku, który jest czytany / zapisywany, ale jeśli nie poda tej nazwy (od razu naciśnie enter), to używana jest domyślna nazwa pliku, np. sklep.csv. Przygotuj taki przykładowy plik z kilkoma produktami.

**Zadanie 2.6 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 2.7 Liczba słownie**

To takie „wyzwanie”, może to zająć sporo czasu i nie musicie tego robić od razu na następne zajęcia.

Stwórz moduł (plik Pythona), w którym zdefiniowana będzie funkcja zamieniająca podaną liczbę całkowitą na postać słowną, np.:

```
>>> liczba_slownie(113)
```

```
'sto trzynaście'
```

Obsłuż jak największy zakres liczbowy, może tryliardy?...

Moduł może zawierać dodatkowe pomocnicze definicje.

Opcjonalnie dodaj także drugą funkcję zwracającą tekst mówiący o kwocie pieniężnej, tak jak umieszcza się ją np. na umowach czy fakturach, np.:

```
>>> kwota_slownie(204)
```

```
'dwieście cztery złote'
```

Napisz testy jednostkowe oraz program interaktywny. Dla chętnych, na dalszym etapie kursu: program okienkowy / prosta aplikacja webowa w Django udostępniające opisaną funkcjonalność.

**Zadanie 2.8 Prognoza pogody**

Usługa [open-meteo.com](https://open-meteo.com) udostępnia dane pogodowe (stan bieżący oraz prognozę) w formacie JSON bez potrzeby rejestracji konta (czego wymaga większość takich serwisów).

Zapytanie pod adres <https://geocoding-api.open-meteo.com/v1/search?name={miasto}> zwraca listę znalezionych lokalizacji o podanej nazwie (lub w których nazwie jest podany fragment). Zwykle pierwsza lokalizacja jest tą najwłaściwszą. Dane lokalizacji obejmują m.in. długość i szerokość geograficzną.

Zapytanie pod adres <https://api.open-meteo.com/v1/forecast?latitude={lat}&longitude={lon}> zwraca dane pogodowe dla podanych współrzędnych geograficznych. Dane obejmują bieżącą pogodę oraz listę prognoz. Sami zbadajcie strukturę wynikowego JSONa - jest trochę bardziej skomplikowana, niż w przypadku walut.

Dodatkowo w zapytaniu można precyzować, czy chcemy dostać bieżącą pogodę, oraz jakie [parametry pogodowe](#) chcemy otrzymać w prognozie godzinnej. Przykładowe zapytanie dla przybliżonych współrzędnych Warszawy:

[https://api.open-meteo.com/v1/forecast?latitude=52.23&longitude=23&current\\_weather=true&hourly=temperature\\_2m,relativehumidity\\_2m,windspeed\\_10m](https://api.open-meteo.com/v1/forecast?latitude=52.23&longitude=23&current_weather=true&hourly=temperature_2m,relativehumidity_2m,windspeed_10m)

Napisz program, który:

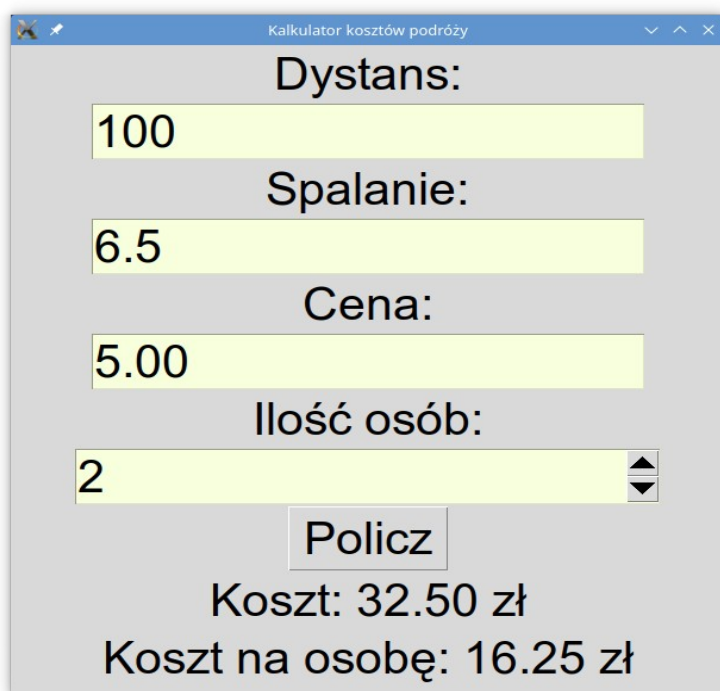
- pobiera od użytkownika nazwę miasta / fragment nazwy lokalizacji
- za pomocą pierwszego zapytania pobiera listę lokalizacji i wypisuje je ponumerowane, dla każdej z nich takie pola: name, country, population, longitude, latitude, elevation
- użytkownik wybiera jedną z nich (podaje numer pozycji na liście)
- program za pomocą drugiego zapytania pobiera dane pogodowe dla wybranej lokalizacji
- wypisuje kilka wybranych parametrów dla bieżącego stanu pogody
- wypisuje prognozę temperatury (pole temperature\_2m) wraz z datami i godzinami, dla których są prognozowane.

Jeśli macie własny pomysł, jak zmienić / uatrakcyjnić taką aplikację, to kombinujcie po swojemu.

**Zadanie 2.9 Kalkulator kosztów podróży w tkinter**

W technologii tkinter napisz aplikację okienkową, w której do 3 pól tekstowych (Entry) wpisuje się: cenę paliwa, średnie spalanie (l/100km) oraz dystans w km, a program oblicza koszt podróży samochodem (spalinowym ;-)).

Poniżej screen przykładowego rozwiązania, w którym dodatkowo za pomocą komponentu Spinbox pobieram info o liczbie osób, na którą mają być podzielone koszty.

**Zadanie 2.10 Pobieranie danych tkinter**

Napisz w technologii tkinter program, który działa w oparciu o dane pobrane z sieci. Może to być:

- Przelicznik walut, który pobiera dane z serwisu NBP, pozwala wybrać walutę (można użyć tk.ttk.Combobox), wpisać kwotę i przeliczyć na PLN lub z PLN.
- Informacje o pogodzie, gdzie można wpisać nazwę lokalizacji, a program wysyłając odpowiednie zapytanie pobierze info o lokalizacji oraz pogodzie i część tych danych wyświetli się w oknie.