**Słowniki i zbiory w języku Python. Serializacja i deserializacja.**

**Słownik** (ang. *dictionary*) jest obiektem przechowującym nieuporządkowaną kolekcję danych, gdzie każdy element składa się z dwóch części: **klucza** i **wartości**. Słowniki w innych językach występują czasami pod nazwą „pamięć asocjacyjna" albo „tablica asocjacyjne". Elementy słownika bardzo często nazywane są **parą klucz-wartość**. Pary te zwykle są określane mianem mapowania (ang. *mapping*), ponieważ każdy klucz jest mapowany na wartość.

Kiedy chcemy ze słownika pobrać określoną wartość, używamy klucza powiązanego z tą wartością. Przypomina to proces wyszukiwania określonego hasła (słowa) w słowniku, w którym słowa są kluczami, natomiast definicje — wartościami. Od Pythona wersji 3.7 mamy gwarancję, że klucze w słowniku będą uporządkowane w kolejności wstawiania ich do słownika, zaś same słowniki obsługują zagnieżdżanie obiektów na dowolną głębokość.

Słownik tworzymy przez umieszczenie zbioru elementów w nawiasie klamrowym. Każdy taki element składa się z klucza, dwukropka i wartości. Poszczególne elementy są rozdzielone przecinkami. Oto przykład prostego słownika:

```
slownik = {'Wiktoria': '500100200', 'Juliusz':'501101201',
'Karina':'503111200',  'Krystian': '500300200'}
```

Polecenie to tworzy słownik i przypisuje mu nazwę `slownik`. Nasz słownik składa się z czterech elementów. Pierwszy element to `'Wiktoria':'500100200'`. W tym elemencie kluczem jest `Wiktoria`, a wartością numer `500100200`. W przykładzie klucze i wartości słownika to ciągi tekstowe. Wartość słownika może być jednak obiektem dowolnego typu, natomiast kluczem musi być obiekt niemodyfikowalny (np. nie może to być lista).

Słowniki znajdują różne zastosowania. Przykładowo, każdy pracownik firmy ma numer ID, a zadanie programisty polega na utworzeniu programu wyszukującego imię (czy nazwisko) pracownika na podstawie ID. Można w tym celu utworzyć słownik, w którym każdy element będzie zawierał klucz w postaci ID i wartość w postaci imienia pracownika. Znając ID pracownika można pobrać jego imię. Odmiennym przykładem może być program pozwalający użytkownikowi na podanie imienia (lub nazwiska) osoby i wyświetlający jej numer telefonu. Taki program mógłby wykorzystać słownik, w którym każdy element zawiera klucz w postaci imienia (nazwiska) osoby oraz wartość w postaci numeru jego telefonu. Znając imię (nazwisko) osoby łatwo otrzymać jego numer telefonu.

**Ćwiczenie 1.**
Napisz program, który tworzy słownik składający się z 4 elementów i drukuje go na ekranie komputera.

```
def main(): #Definicja funkcji o nazwie main()
    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'}
    print("Wydruk słownika: ", slownik, sep = "")
main() #Wywołanie funkcji main()
```

**Ćwiczenie 2.**
Na podstawie **Ćwiczenia 1** napisz program, który dodaje element do istniejącego słownika i usuwa z niego stary element. Dodatkowo program ma podać, ile elementów było przed każdą z operacji.

```
def main():
    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'}
```

```
    print("Zawartość słownika: ", slownik, ".", sep = "")
    print()
    print("W słowniku znajdują się", len(slownik), "elementy.")
    print()

    slownik['Katarzyna'] = '400100200' #Dodanie do słownika nowego
    elementu
    print("Zawartość słownika po dodaniu nowego elementu: ", slownik,
    ".", sep = "")
    print()
    print("W słowniku znajduje się", len(slownik), "elementów.")
    print()

    del slownik['Wiktoria'] #Usunięcie ze słownika starego elementu
    print("Zawartość słownika po usunięciu elementu: ", slownik, ".",
    sep = "")
    print()
    print("W słowniku znajdują się", len(slownik), "elementy.")
main()
```

W poniższej tabeli zawarte są popularne metody związane ze słownikami.

**Wybrane metody słownika**

| Metoda | Opis |
|---|---|
| `clear()` | Usuwa zawartość słownika |
| `get()` | Pobiera wartość danego klucza. Jeśli klucz nie zostanie znaleziony, metoda nie zgłasza wyjątku, lecz zwraca wartość domyślną |
| `items()` | Zwraca wszystkie pary `klucz-wartość` słownika w postaci sekwencji krotek |
| `keys()` | Zwraca wszystkie klucze słownika w postaci sekwencji krotek |
| `pop()` | Usuwa klucz ze słownika i jednocześnie zwraca jego wartość. Jeśli klucz nie zostanie znaleziony, metoda zwraca wartość domyślną |
| `values()` | Zwraca wszystkie wartości słownika w postaci sekwencji krotek |

**Ćwiczenie 3.**
Napisz program, który korzystając z metody `clear()`, usuwa całą zawartość słownika.

```
def main():
    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'}
    print("Zawartość słownika: ", slownik)
    print()
    print("W słowniku znajdują się", len(slownik), "elementy.")
    print()

    slownik.clear() #Usunięcie zawartości słownika
    print("Zawartość słownika po zastosowaniu metody clear(): ")
    print()
    print(slownik) #Wydruk pustego słownika
    print()
    print("W słowniku znajduje się", len(slownik), "elementów.")
main()
```

**Ćwiczenie 4.**
Napisz program, który demonstruje działanie metody `get()`.

```
def main():
    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'}
    print("Zawartość słownika: ", slownik)
    print("Zawartość słownika: ", slownik, ".", sep = "")
    print()

    wartosc = slownik.get('Wiktoria', 'Element nie został
    znaleziony.')
    print(wartosc)
    print()

    wartosc = slownik.get('Krzysztof', 'Element nie został
    znaleziony.')
    print(wartosc)
main()
```

**Ćwiczenie 5.**
Napisz program, który zwraca wszystkie klucze słownika i ich wartości w postaci sekwencji krotek.

```
def main():
    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'}
    print("Zawartość słownika: ", slownik)
    print()

    print("Sekwencja krotek zawierająca pary klucz-wartość
    słownika:")
    print()
    for klucz, wartosc in slownik.items():
        print(klucz, wartosc)
main()
```

**Ćwiczenie 6.**
Napisz program, który demonstruje działanie metody `keys()`.

```
def main():
    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'}
    print("Zawartość słownika: ", slownik)
    print()

    print("Wynikiem wywołania metody keys() jest sekwencja: ")
    print()
    for klucz in slownik.keys():
        print(klucz)
main()
```

**Ćwiczenie 6.**
Napisz program, który demonstruje działanie metody `pop()`.

```
def main():
    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'}
    print("Zawartość słownika: ", slownik)
    print()

    print("Wynik wywołania metody pop(): ")
    print()
    slownik_num = slownik.pop("Wiktoria", "Element nie został
    znaleziony")
    print(slownik_num)
    print()

    print("Zawartość słownika: ", slownik, sep = "")
    slownik_num = slownik.pop("Jacek", "Element nie został
    znaleziony")
    print()
    print(slownik_num)
    print()
    print("Zawartość słownika: ", slownik,  sep = "")
main()
```

**Ćwiczenie 7.**
Napisz program, który demonstruje działanie metody `values()`.

```
def main():
    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'}
    print("Zawartość słownika: ", slownik)
    print()

    print("Wynikiem wywołania metody values() jest następująca
    sekwencja: ")
    print()
    for val in slownik.values():
        print(val)
main()
```

**Zbiorem** (ang. *set*) nazywamy kolekcję unikatowych wartości – żadne elementy nie mogą takiej samej wartości, a jego działanie podobne jest do zbioru w matematyce. Zbiór tworzymy wywołując wbudowaną funkcję `set()`.

Zbiór jest nieuporządkowany, zaś jego elementy nie są przechowywane w żadnej konkretnej kolejności, przy czym typy danych w zbiorze mogą być różne. Zbiór jest modyfikowalny, ponieważ na zbiorach można wykonywać różne operacje, w tym dodawanie i usuwanie elementów, a ponadto możliwe jest uzyskanie sumy, różnicy oraz iloczynu zbiorów. Da się również uzyskać różnicę symetryczną.

**Ćwiczenie 8.**
Korzystając z funkcji `add()` oraz `update()` napisz program, który do pustego zbioru o nazwie `zbior` dodaje nowe elementy i usuwa z niego stare. Dodatkowo wyznacz liczbę tych elementów w zbiorze.

```
def main():
    zbior = set() #Tworzenie pustego zbioru
    print()
    print("Liczba elementów w zbiorze = ", len(zbior), sep = "")
    print()
    print("Dodajemy nowe elementy:")
    print()
    print("Dodano 1. element ")
    zbior.add(1) #Dodanie jednego elementu o wartości 1

    print("Liczba elementów w zbiorze = ", len(zbior), sep = "")
    print("Dodano 2. element ")
    zbior.add(2)

    print("Liczba elementów w zbiorze = ", len(zbior), sep = "")

    print("Dodano cztery elementy ")
    zbior.update([3, 4, 5, 6]) #Dodanie 4 nowych elementów

    print("Liczba elementów w zbiorze = ", len(zbior), sep = "")

    print()
    print("Oto nasz cały zbiór: ", zbior, sep = "")
    print()
    print("Teraz skasujemy nasz zbiór!")
    print()
    zbior.clear() #Kasowanie całego zbioru
    print("Liczba elementów w zbiorze = ", len(zbior), sep = "")
main()
```

**Ćwiczenie 9.**
Napisz program, który za pomocą metody `union()` znajduje sumę dwóch zbiorów.

```
def main():
    zbior1 = set([1, 2, 3, 4, 5]) #Tworzenie pierwszego zbioru
    zbior2 = set([5.5, 6.5, 7.5, 8.5]) #Tworzenie drugiego zbioru
    print("Zbiór 1 = ", zbior1, sep = "")
    print("Zbiór 2 = ", zbior2, sep = "")
    zbior = zbior1.union(zbior2) #Tworzenie zbioru wynikowego,
    zawierającego sumę dwóch zbiorów
    print()
    print("Suma dwóch zbiorów = ", zbior, sep = "")
main()
```

**Uwaga:**
Do tego samego wyniku prowadzi użycie instrukcji: `zbior = zbior1 | zbior2`.

**Ćwiczenie 10.**
Napisz program ilustrujący różnicę dwóch zbiorów za pomocą metody `difference()`. Sprawdź, czy do tego samego wyniku prowadzi użycie operatora − (minus).

```
def main():
    zbior1 = set([1, 2, 3, 4.5, 5.5]) #Tworzenie pierwszego zbioru
    zbior2 = set([1, 5.5, 6.5, 7.5, 8.5]) #Tworzenie drugiego zbioru
    print("Zbiór 1 = ", zbior1, sep = "")
    print("Zbiór 2 = ", zbior2, sep = "")
    zbior = zbior1.difference(zbior2) #Tworzenie zbioru wynikowego,
    zawierającego różnicę dwóch zbiorów
    print()
    print("Różnica dwóch zbiorów = ", zbior, sep = "")
main()
```

**Ćwiczenie 11.**
Napisz program ilustrujący iloczyn (część wspólną) dwóch zbiorów za pomocą metody `intersection()`. Sprawdź, czy do tego samego wyniku wystarczy użycie operatora `&` (ampersand).

```
def main():
    zbior1 = set([1, 2, 3, 4.5, 5.5]) #Tworzenie pierwszego zbioru
    zbior2 = set([1, 5.5, 6.5, 7.5, 8.5]) #Tworzenie drugiego zbioru
    print("Zbiór 1 = ", zbior1, sep = "")
    print("Zbiór 2 = ", zbior2, sep = "")
    zbior = zbior1.intersection(zbior2) #Tworzenie zbioru wynikowego,
    zawierającego iloczyn dwóch zbiorów
    print()
    print("Iloczyn dwóch zbiorów = ", zbior, sep = "")
main()
```

**Ćwiczenie 12.**
Napisz program ilustrujący różnicę symetryczną dwóch zbiorów za pomocą metody `symmetric_difference()`. Sprawdź, czy do tego samego wyniku prowadzi użycie operatora ^ (caret).

```
def main():
    zbior1 = set([1, 2, 3, 4.5, 5.5]) #Tworzenie pierwszego zbioru
    zbior2 = set([1, 5.5, 6.5, 7.5, 8.5]) #Tworzenie drugiego zbioru
    print("Zbiór 1 = ", zbior1, sep = "")
    print("Zbiór 2 = ", zbior2, sep = "")
    zbior = zbior1.symmetric_difference(zbior2) #Tworzenie zbioru
    wynikowego, zawierającego różnicę symetryczną dwóch zbiorów
    print()
    print("Różnica symetryczna dwóch zbiorów = ", zbior, sep = "")
main()
```

**Serializacją** (ang. *serialization*) w programowaniu nazywamy proces przekształcania (konwersji) obiektów do strumienia bajtów z zachowaniem ich aktualnego stanu, w celu zapisania w pamięci dyskowej ewentualnie przesłania obiektu do innego procesu albo do innego komputera. Natomiast **deserializacja** oznacza proces odwrotny. W języku Python procesy takie noszą nazwę `picklingu` (od modułu `pickle`, który zawiera wiele funkcji przeznaczonych do realizowania serializacji obiektów).

Spis czynności, które należy wykonać krok po kroku, aby dokonać serializacji obiektu w języku Python, jest następujący:

- zaimportować moduł `pickle`,
- otworzyć plik w trybie zapisu binarnego,
- wywołać metodę `dump()` z modułu `pickle` i zapisać obiekt w podanym pliku,
- zamknąć plik po zapisaniu wszystkich obiektów.

**Ćwiczenie 13.**
Napisz program ilustrujący serializację / deserializację słownika do/z pliku binarnego `info.dat`.

```
import pickle

def main():
    print("Program przedstawia proces serializacji i deserializacji
    obiektu")
    print()

    koniec_pliku = False #Zmienna pomocnicza wskazująca koniec pliku

    slownik = {'Wiktoria':'500100200', 'Juliusz':'501101201',
    'Karina':'503111200', 'Krystian':'500300200'} #Utworzenie
    słownika i przypisanie go do zmiennej slownik
    print("Zawartość słownika: ", slownik, sep = "")
    print()

    print("Teraz nastąpi serializacja słownika do pliku info.dat")
    plik_wejsciowy = open("info.dat", "wb") #Otwarcie pliku w trybie
    zapisu binarnego
    pickle.dump(slownik, plik_wejsciowy) #Serializacja słownika
    plik_wejsciowy.close() #Zamknięcie pliku
    print()

    print("W tym momencie możemy zacząć deserializację słownika")
    print()
    plik_wyjsciowy = open("info.dat", "rb") #Otwarcie pliku w trybie
    odczytu binarnego

    while not koniec_pliku:
        try:
            slownik1 = pickle.load(plik_wyjsciowy)
            print("Wydruk z pliku binarnego: ", slownik1, sep = "")
        except EOFError:
            koniec_pliku = True
    plik_wyjsciowy.close() #Zamknięcie pliku binarnego
main()
```

**Uwaga:**
W programie zastosowano obsługę wyjątków. Kiedy Python napotyka błąd, zatrzymuje bieżący program i wyświetla komunikat o błędzie, czyli faktycznie zgłasza tzw. wyjątek wskazując, że zdarzyło się coś niespodziewanego. Jeśli programista nic z tym nie zrobi, Python przerywa działanie i wyświetla szczegóły tego wyjątku. Najbardziej elementarnym sposobem obsługi (wyłapywania) wyjątków jest użycie instrukcji `try` z klauzulą `except`. Dzięki użyciu instrukcji `try` wydzielony zostaje pewien fragment kodu, który mógłby potencjalnie wywołać wyjątek. Następnie tworzy się klauzulę `except` z blokiem instrukcji, które są wykonywane tylko w przypadku zgłoszenia wyjątku.

## Zadania domowe

**Zadanie 1.**
Liczbami półpierwszymi nazywamy liczby naturalne, które są iloczynem dwóch liczb pierwszych. Przykładowo liczbami taki są: $34$, bo $34 = 2 \cdot 17$, a także $699 = 2 \cdot 17$, $841 = 29 \cdot 29$ itd.

W pliku `liczby.txt` znajduje się 500 liczb naturalnych, z których każda ma co najwyżej 6 cyfr. Napisz program, który wyodrębni wszystkie liczby półpierwsze z pliku `liczby.txt`. Program powinien zapisać wyodrębnione liczby do pliku `liczby_wynik.txt`.

**Zadanie 2.**
W pliku `dane.txt` znajduje się 200 wierszy. Każdy wiersz zawiera 320 liczb naturalnych z przedziału od 0 do 255, oddzielonych znakami pojedynczego odstępu (spacjami). Przedstawiają one jasności kolejnych pikseli czarno-białego obrazu o wymiarach 320 x 200 pikseli (0 – czarny, 255 – biały). Napisz program, który:

1. Podaje jasność najjaśniejszego i najciemniejszego piksela.
2. Podaje, ile wynosi najmniejsza liczba wierszy, które należy usunąć, żeby obraz miał pionową oś symetrii. Obraz ma pionową oś symetrii, jeśli w każdym wierszu $i - ty$ piksel od lewej strony przyjmuje tę samą wartość, co $i - ty$ piksel od prawej strony, dla dowolnego $1 \leq i \leq 320$.
3. Podaje liczbę wszystkich pikseli kontrastujących. Dwa sąsiednie piksele nazywamy kontrastującymi, jeśli ich wartości różnią się więcej niż 128, zaś sąsiednimi nazywamy takie piksele, które leżą obok siebie w tym samym wierszu lub w tej samej kolumnie.
4. Zapisuje wyniki z punktów od 1 do 3 w pliku `dane_wyniki.txt`.

Termin przysłania rozwiązanych zadań mija w dniu **18 maja 2022 r.** (o północy). Proszę pamiętać, aby w nazwie pliku z programem była data dzisiejszych zajęć, czyli 12.05.2022.