Pojemniki na dane – Listy, Krotki, Zbiory…

Agenda:

Część 1: Podstawowe Kolekcje

- Lista (list): Dynamiczna lista zakupów.
- Krotka (tuple): Niezmienny strażnik danych.
- Zbiór (set): Specjalista od unikalności.
- Pułapka haszowalności: TypeError: unhashable type.
- Zadania praktyczne: "Analizator tagów" i "Wspólni znajomi".

Część 2: Najważniejsza Struktura Danych - Słownik (dict)

- Problem: Potrzebujemy etykiet dla naszych danych.
- Anatomia słownika: Pary klucz: wartość.
- Ćwiczenie 1: Nasza pierwsza wizytówka w słowniku.
- Operacje CRUD: Dodawanie, odczyt, aktualizacja i usuwanie.
- Ćwiczenie 2: Dynamiczna książka kontaktowa.
- Iteracja po słowniku: .keys(), .values(), .items().
- Ćwiczenie 3: Ładne wyświetlanie książki kontaktowej.

Część 3: Złożone Struktury Danych

- Zagnieżdżanie: Prawdziwa potęga struktur danych.
- Najczęstszy wzorzec: Lista słowników.
- Zadanie końcowe: Analiza "Bazy Danych Pracowników".

Przypomnienie: Nasze skrypty mają już "układ nerwowy"

Co potrafią nasze pythonowe skrypty?

- **Myśleć**: Podejmować decyzje za pomocą if-elif-else.
- **Być cierpliwym:** Powtarzać zadania w pętlach **while i for**.
- **Sterować logiką**: Używać **and, or, not, break, continue**.

Ograniczenie: Każda zmienna (**suma_parzystych, wejscie, liczba**) przechowuje tylko jedną wartość naraz.

```
# Program, który prosi o liczby, dopóki nie wpiszesz "koniec",
# a następnie sumuje tylko te parzyste.
suma_parzystych = 0
while True:
    wejscie = input("Podaj liczbę (lub 'koniec'): ")
    if wejscie.lower() == 'koniec':
        break

    liczba = int(wejscie)
    if liczba % 2 != 0: # Jeśli nieparzysta
        continue      # Pomiń i weź następną

    suma_parzystych += liczba

print(f"Suma podanych liczb parzystych to: {suma_parzystych}")
```

Projektujemy dynamiczną listę

Problem: Jak przechować wiele wartości w jednej zmiennej?

Rozwiązanie: Lista (list) - uporządkowana, zmienna (mutowalna) kolekcja elementów.

Składnia: Nawiasy kwadratowe **[]** , elementy oddzielone przecinkami.

```
pusta_lista = []
imiona = ["Anna", "Piotr", "Zofia"]
```

Projekt Ćwiczenia 1: "Dynamiczna lista zakupów"

Cel: Stworzyć program, który pozwala użytkownikowi dodawać produkty do listy zakupów, a na koniec ją wyświetla.

Pseudokod:

START

UTWÓRZ pustą listę o nazwie "zakupy"

DOPÓKI PRAWDA:

WYŚWIETL "Podaj produkt (lub 'koniec'): "

POBIERZ produkt

JEŚLI produkt == "koniec" TO

PRZERWIJ pętlę

DODAJ produkt DO listy "zakupy"

WYŚWIETL "Twoja lista zakupów:"

WYŚWIETL lista "zakupy"

STOP

Ćwiczenie 1: Implementacja w Pythonie

Cel: Przełożenie projektu (schematu i pseudokodu) na działający kod.

Nowe narzędzie: Metoda **.append()**

lista.append(element) - dodaje **element** na sam koniec listy.

Ulepszamy Listę: Sortowanie i prezentacja danych

Cel: Ulepszyć program "Lista Zakupów", aby sortował produkty i wyświetlał je w czytelny, ponumerowany sposób.

Nowe narzędzia:

- **lista.sort():** Sortuje listę alfabetycznie w miejscu (in-place).
- **len(lista):** Zwraca liczbę elementów.
- **enumerate(lista):** Daje nam w pętli indeks i wartość.

Ćwiczenie 2: “Bezpieczne współrzędne”

Problem: Co, jeśli chcemy mieć pewność, że dane nigdy się nie zmienią po utworzeniu? (np. współrzędne, kolor RGB, PESEL)

Rozwiązanie: Krotka (**tuple**) - uporządkowana, **NIEZMIENNA (niemutowalna)** kolekcja.

Cel: Zobaczyć w praktyce, jak krotka chroni dane przed modyfikacją.

Składnia: Nawiasy okrągłe **().**

Zadanie:

a. Stwórz krotkę punkt_startowy = (0, 0).

b. Wyświetl jej zawartość.

c. Spróbuj zmienić jej pierwszą wartość: punkt_startowy[0] = 10.

d. Zaobserwuj i przeanalizuj błąd.

```
wspolrzedne = (10, 20)
kolor_czerwony = (255, 0, 0)

# UWAGA: Krotka z jednym elementem wymaga przecinka!
krotka_jednoelementowa = (1,)
```

Ćwiczenie 3 – Specjalista od unikalności: Zbiór (set)

Problem: Jak łatwo usunąć duplikaty z listy? Jak przechowywać tylko unikalne wartości?

Rozwiązanie: Zbiór (**set**) - nieuporządkowana, mutowalna kolekcja unikalnych, haszowalnych elementów.

Cel: Zobaczyć w praktyce, jak zbiór upraszcza usuwanie powtórzeń.

Zadanie:

a. Stwórz listę *numery_lotto = [5, 12, 5, 42, 12, 9]*.

b. Przekonwertuj listę na zbiór, aby usunąć duplikaty.

c. Wyświetl wynikowy zbiór.

d. Przekonwertuj zbiór z powrotem

```
liczby = {1, 2, 3, 3, 2, 1}
print(liczby) # -> {1, 2, 3}

# UWAGA: Pusty zbiór tworzymy TYLKO za pomocą set()!
pusty_zbior = set()
# pusty_slownik = {} # {} tworzy pusty słownik!
```

Pułapka niemutowalności i TypeError: unhashable type

Problem: Co się stanie, gdy spróbujemy dodać listę do zbioru?

```
moj_zbior = {1, 2, 3}
# moj_zbior.add([4, 5])
# -> TypeError: unhashable type: 'list'
```

A co z krotką, która zawiera listę?

```
krotka_z_lista = (1, 2, [3, 4])
# moj_zbior.add(krotka_z_lista)
# -> TypeError: unhashable type: 'list'
```

Wyjaśnienie: Haszowalność (Hashability)

- **Analogia:** Hasz to unikalny, stały "odcisk palca" obiektu.
- Zbiory i klucze słowników używają haszy do znajdowania elementów.
- **Złota zasada:** Obiekt jest haszowalny, jeśli jego wartość nigdy się nie zmienia.
- Lista nie jest haszowalna, bo można ją zmienić. Jej "odcisk palca" nie byłby stały.
- Krotka jest haszowalna, **tylko jeśli** wszystkie jej elementy też są haszowalne. Nasza krotka zawierała listę, więc cała stała się "niehaszowalna".

Zadanie do przemyślenia: Dlaczego **frozenset** jest haszowalny, a **set** nie?

Zadanie 1 – Analizator tagów

Cel: Przetworzenie i analiza danych tekstowych z użyciem list i zbiorów.

Problem: Napisz program, który prosi użytkownika o listę tagów (słów kluczowych) oddzielonych przecinkami. Program ma przeanalizować te tagi i wyświetlić podsumowanie.

Przykład wejścia: python, programowanie, python, kurs, nauka, kurs

Kroki do wykonania:

- Pobierz od użytkownika jeden ciąg znaków z tagami.
- Użyj metody **.split(',')**, aby zamienić string na listę tagów.
- Wyczyść dane: Stwórz nową listę, w której każdy tag będzie pozbawiony białych znaków z początku i końca (użyj metody **.strip()**).
- Użyj zbioru (set), aby znaleźć unikalne tagi.
- Wyświetl, ile było wszystkich tagów, a ile unikalnych.
- Wyświetl unikalne tagi w porządku alfabetycznym (użyj sorted()).

```
Liczba wszystkich podanych tagów: 6
Liczba unikalnych tagów: 4

Unikalne tagi (alfabetycznie):
- kurs
- nauka
- programowanie
- python
```

Zadanie 2 – Wspólni znajomi

Cel: Zastosowanie zbiorów do efektywnego znajdowania części wspólnej dwóch kolekcji.

Problem: Masz dwie listy znajomych. Znajdź osoby, które są na obu listach.

Nowe narzędzia: Operacje na zbiorach

- **set(lista):** Konwertuje listę na zbiór, automatycznie usuwając duplikaty.
- **zbior1.intersection(zbior2):** Zwraca nowy zbiór zawierający elementy wspólne dla **zbior1** i **zbior2**.
- **zbior1 & zbior2:** Operator przecięcia - krótszy i bardziej czytelny sposób na to samo co **.intersection().**

Kroki do wykonania:

a. Przekonwertuj obie listy znajomych na zbiory.

b. Użyj operatora **&** lub metody **.intersection(),** aby znaleźć wspólne elementy.

c. Wyświetl wynik.

```
znajomi_anny = ["Piotr", "Zofia", "Marek", "Anna"]
znajomi_piotra = ["Anna", "Marek", "Krzysztof", "Ewa"]
```

Omówienie Zadania 2 – “Wspólni znajomi”

Cel: Analiza rozwiązania i utrwalenie potęgi zbiorów w operacjach matematycznych.

Kluczowa koncepcja: Użycie odpowiedniej struktury danych (**set**) drastycznie upraszcza kod i poprawia jego wydajność.

Słowniki (dict)

Problem: Listy i krotki używają indeksów (**0, 1, 2**...). Musimy pamiętać, co jest pod którym numerem.

- **osoba = ["Jan", "Kowalski", 45]** - Co oznacza **osoba[2]**?

Rozwiązanie: Słownik (dict) - kolekcja par **klucz: wartość.**

- **Zamiast indeksów, używamy opisowych etykiet (kluczy).**
- **osoba = {"imie": "Jan", "nazwisko": "Kowalski", "wiek": 45}**
- **Dostęp do danych jest czytelny: osoba["wiek"]**

Ćwiczenie 4 - Nasza pierwsza wizytówka w słowniku

Problem: Jak przechowywać dane, które mają opisowe etykiety?

- Lista: **["Jan", "Kowalski", 45]** - Co oznacza [**2**]? Musimy pamiętać.

Rozwiązanie: Słownik **(dict)** – kolekcja par **klucz: wartość.**

Składnia:

- Nawiasy klamrowe **{}.**
- Klucze muszą być **unikalne i niemutowalne** (najczęściej **str**).

Cel: Stworzyć słownik i odczytać z niego dane po kluczach.

Zadanie:

a. Stwórz słownik **wizytowka** z kluczami: **imie, nazwisko,stanowisko.**

b. Wypełnij go swoimi danymi.

c. Wyświetl imię i nazwisko w jednej linii, pobierając je ze słownika.

d. W kolejnej linii wyświetl stanowisko.

```
osoba = {
    "imie": "Jan",
    "nazwisko": "Kowalski",
    "wiek": 45
}

print(f"Imię: {osoba['imie']}") # ->
Imię: Jan
```

Modyfikowanie Słownika: Dodawanie, Zmiana, Usuwanie

Cel: Zrozumieć, jak dynamicznie zmieniać zawartość słownika.

1. Dodawanie nowej pary / Aktualizacja istniejącej:

- Używamy tej samej składni **slownik[klucz] = wartosc.**
- Jeśli **klucz** nie istnieje, zostanie utworzony.
- Jeśli **klucz** istnieje, jego wartość zostanie nadpisana.

2. Usuwanie pary klucz: wartość:

- Używamy słowa kluczowego **del:** **del slownik[klucz] .**

Zadanie 5: Dynamiczna książka kontaktowa

Cel: Samodzielne stworzenie interaktywnego programu, który zarządza kolekcją danych w słowniku.

Problem: Napisz program, który działa jak prosta książka kontaktowa.

Logika programu:

- Stwórz pusty słownik **kontakty = {}.**
- W pętli **while True** wyświetlaj menu i proś użytkownika o wybór opcji.
- Użyj **if/elif/else**, aby obsłużyć wybór użytkownika.

Menu opcji:

- **Dodaj kontakt** (pyta o nazwę i numer, dodaje do słownika)
- **Wyświetl kontakt** (pyta o nazwę, wyświetla numer lub komunikat o braku)
- **Usuń kontakt** (pyta o nazwę, usuwa ze słownika)
- **Wyświetl wszystko** (na razie wystarczy **print(kontakty))**
- **Zakończ** (przerywa pętlę)

Wskazówki:

- Jak sprawdzić, czy kontakt już istnieje, zanim go wyświetlisz lub usuniesz? Użyj operatora **in: if nazwa in kontakty**:.
- Pamiętaj o obsłudze przypadku, gdy użytkownik prosi o kontakt, którego nie ma w książce.

Omówienie Zadania 5: Dynamiczna książka kontaktowa

Cel: Analiza kompletnego rozwiązania i utrwalenie wzorca pętli sterowanej przez użytkownika.

Iteracja po słowniku: .keys(), .values(), .items()

Problem: W poprzednim ćwiczeniu **print(kontakty)** wyświetlało dane w technicznej, nieczytelnej formie. Jak to zrobić ładnie?

Rozwiązanie: Użycie pętli **for** ze specjalnymi metodami słownika.

Metody iteracji:

- **for klucz in slownik.keys():** - iteruje po samych **kluczach**.
- **for wartosc in slownik.values():** - iteruje po samych **wartościach**.
- **for klucz, wartosc in slownik.items():** - iteruje po parach **(klucz, wartość)** (najczęściej używany sposób).

Ćwiczenie 6 - Wyświetlanie książki kontaktowej

Cel: Ulepszyć opcję "Wyświetl wszystko" w programie "Książka kontaktowa", aby prezentowała dane w czytelny sposób.

Problem: Obecnie **print(kontakty)** wyświetla słownik w surowej formie. Zmieńmy to!

Zadanie:

a. Otwórz swój program "Dynamiczna książka kontaktowa" (Ćwiczenie 5).

b. Znajdź blok kodu odpowiedzialny za opcję 4. Wyświetl wszystko.

c. Zastąp print(kontakty) pętlą for, która będzie iterować po parach klucz: wartość w słowniku kontakty.

d. Dla każdej pary wyświetl kontakt w formacie: Nazwa: [nazwa_kontaktu], Numer: [numer_telefonu].

e. Dodaj nagłówek i stopkę, aby wynik był estetyczny (np. --- MOJE KONTAKTY ---).

f. Wskazówka: Użyj metody .items() i f-stringów.

```
--- MOJE KONTAKTY ---
Nazwa: Anna, Numer: 123456789
Nazwa: Piotr, Numer: 987654321
--- KONIEC LISTY ---
```

Omówienie Ćwiczenia 6

Cel: Analiza poprawnego sposobu iteracji po słowniku i formatowania wyniku.

Kluczowe techniki:

- Użycie **if not kontakty:** do obsługi przypadku pustego słownika.
- Użycie **kontakty.items()** do eleganckiej iteracji po kluczach i wartościach jednocześnie.
- Użycie **f-stringów** do czytelnego formatowania wyjścia.

Zagnieżdżanie: Prawdziwa potęga struktur danych

Problem: Słownik jest świetny do opisu jednego obiektu (np. jednej osoby). Jak przechować listę takich obiektów?

Rozwiązanie: Zagnieżdżanie! Możemy tworzyć listy słowników, słowniki list, słowniki słowników...

Najczęstszy wzorzec: Lista słowników

Dostęp do danych zagnieżdżonych:

- **uzytkownicy[0]** -> Zwraca pierwszy słownik: {"imie": "Anna", ...}
- **uzytkownicy[0]["imie"]** -> Zwraca imię z pierwszego słownika: "Anna"

```
uzytkownicy = [
    {"imie": "Anna", "wiek": 28, "miasto": "Gdańsk"},
    {"imie": "Piotr", "wiek": 35, "miasto": "Warszawa"},
    {"imie": "Zofia", "wiek": 22, "miasto": "Kraków"}
]
```

Zadanie 5: Baza Danych Pracowników

Cel: Zastosowanie listy słowników do przechowywania i analizy złożonych danych.

Problem: Masz listę pracowników, gdzie każdy pracownik to słownik. Przeanalizuj te dane.

Zadania do wykonania:

a) Oblicz i wyświetl **średnią pensję** wszystkich pracowników.

b) Znajdź i wyświetl dane pracownika, który zarabia najwięcej.

c) Wyświetl imiona wszystkich osób na stanowisku "Specjalista".

Wskazówka: Użyj pętli **for**, aby przejść po liście. W każdej iteracji będziesz miał dostęp do jednego słownika (pracownika).

Dane wejściowe (kod):

```
baza_danych = [
    {"imie": "Anna", "stanowisko": "Specjalista", "pensja": 4500},
    {"imie": "Piotr", "stanowisko": "Manager", "pensja": 8000},
    {"imie": "Zofia", "stanowisko": "Specjalista", "pensja": 5200},
    {"imie": "Krzysztof", "stanowisko": "Stażysta", "pensja": 2500}
]
```

Omówienie Zadania 5: Baza Danych Pracowników

Cel: Analiza rozwiązania i utrwalenie wzorców przetwarzania listy słowników.

Kluczowe wzorce:

- **Akumulator:** Użycie zmiennej (**suma_pensji**) do zliczania wartości w pętli.
- **Śledzenie maksimum:** Użycie zmiennych do przechowywania aktualnego maksimum i obiektu z nim związanego.
- **Filtrowanie:** Użycie **if** wewnątrz pętli do wybrania tylko pasujących elementów.

Podsumowanie

Co dzisiaj zrobiliśmy?

- Opanowaliśmy 4 fundamentalne struktury danych: list, tuple, set i dict.
- Zrozumieliśmy kluczową różnicę między mutowalnością a niemutowalnością.
- Nauczyliśmy się modelować złożone dane za pomocą listy słowników.

Kluczowa lekcja: Umiejętność wyboru odpowiedniej struktury danych to fundament efektywnego programowania.

Na następnych zajęciach:

- Przestaniemy być tylko użytkownikami narzędzi. Zaczniemy budować **własne**.
- Poznamy **funkcje**, czyli reużywalne bloki kodu.
- Nauczymy się, jak unikać powtarzania kodu (zasada DRY).