

# Struktury danych, typy złożone

- służą do organizowania i przechowywania danych
- jest ich wiele rodzajów. Wybierane są pod kątem konkretnego zastosowania - czy do danego celu dana struktura będzie efektywna i czy umożliwi korzystanie z danych w sposób nie zaskakujący odbiorców (programistów)
- sama w sobie struktura niesie ze sobą znaczenie i spełnia pewne oczekiwania np. wybierając w Pythonie listę programista wie, że może ją modyfikować i że powiązane są z nią pewne metody służące do jej obsługi

## Listy i krotki

W obu wypadkach:

- każdy element ma swój indeks umożliwiający jego szybkie odnalezienie
- obie mogą posiadać duplikaty
- w obu wypadkach można się przemieszczać (iterate) po ich elementach

**ALE...**

## Listy (list)

- są modyfikowalne (mutable) tzn. można do nich dodawać / odejmować nowe elementy
- bardzo często przemieszczamy się w ich wypadku po elementach
- raczej unika się sięgania po elementy umieszczone gdzieś w środku list
- dobrą analogią mogą być książki na półce
- deklaruje się je poprzez użycie nawiasów kwadratowych `[]` np.

```
moje_książki = ["Blade Runner", "Hobbit", "Unicorn Project"]
```

## Krotki (tuple)

- są niemodyfikowalne (immutable)

- ponieważ ich długość jest stała częściej sięgamy po elementy znajdujące się gdzieś głęboko w nich
- dobrym przykładem mogą być np. kategorie w menu restauracji
- deklaruje się je poprzez użycie nawiasów okrągłych `()` np.

```
menu_kategorie = ("przystawki", "zupy", "desery")
```

## Indeksy

Chcąc "wyciągnąć" konkretny element będący ich częścią wpisujemy identyfikator w nawiasach kwadratowych po nazwie zmiennej .

`nazwa_zmiennej[indeks]` np.

```
moja_lista = ["cos1", "cos2", "cos3", "cos4" ]

print(moja_lista[2])
```

WAŻNE!!! Indeksy są liczone od 0 nie od 1 . Stąd też pierwszy element na powyższej liście ma indeks 0, ostatni 3.

Możliwe jest też sięgnięcie "wstecz" np. ostatni element można wskazać przez użycie indeksu -1.

## "Cięcie" list i krotek

Chcąc wyświetlić zakres elementów listy / krotki należy wpisać w nawiasie kwadratowym element startowy i element końcowy oddzielone dwukropkiem.

`moja_zmienna[start:koniec]`

np.

```
print(moja_lista[1:3])
```

Jeśli jest potrzeba "przeskakiwania niektórych wartości to można ten schemat rozszerzyć:

`moja_lista[start:koniec:krok]`

Należy to czytać - "wybierz elementy z moja\_lista, zaczynając od `start`, zatrzymując się PRZED `koniec` (czyli element o tym indeksie nie wejdzie w wybrany zakres) i przeskakując co `skok` .

Co ciekawe przy tych "wycinkach" (slices) list/krotek podanie indeksów poza zakresem etc NIE spowoduje błędu - stworzy po prostu pustą listę/krotkę.

```
lista_a = [0, 1, 2, 3]
wycinek = lista_a[8:12] # to nie spowoduje błędu!
print(wycinek)
# wynikiem będzie:
# []
# ... czyli powstała pusta lista
```

## Metody dla list i krotek

Metody to specjalne funkcje "wbudowane" w dany obiekt. Wywołuje się je poprzez wpisanie nazwy zmiennej, postawienie kropki i wstawienie parametrów do metody w nawiasach okrągłych (jeśli nie przekazujemy żadnych zostawiamy same nawiasy okrągłe).

| krotka   | lista     | opis                                                         |
|----------|-----------|--------------------------------------------------------------|
| .count() | .count()  | zlicza ilość wystąpień przekazanego parametru                |
| .index() | .index()  | podaje pierwszy indeks pod którym jest przekazana wartość    |
|          | .append() | dodaj argument do listy                                      |
|          | .pop()    | usuń element o podanym indeksie z listy(lub ostatni element) |
|          | .sort()   | sortuje elementy                                             |
|          | .copy()   | zwraca kopię obiektu                                         |

## Słowniki i zbiory

Zarówno słowniki jak i zbiory przechowują "klucze", które nie mogą się powtarzać. Jednak w przypadku słowników z kluczem jest powiązana wartość.

Klucze muszą być wartością na której da się wykonać tzw. funkcję skrótu (są "hashable"). W tym wypadku chodzi o to, że muszą to być obiekty, które nie są modyfikowalne (np. ciągi znaków, liczby).

## Słowniki (dict)

Umożliwiają szybkie odnalezienie jakiejś zawartości z pomocą wartości klucza przypisanego do niej.

Deklarujemy je poprzez użycie nawiasów klamrowych, a w nich pary klucz, wartość rozdzielonych dwukropkiem.

```
**moj_słownik={klucz:wartość}**
```

np.

```
menu ={\n    "zupa" : "ogórkowa",\n    "lody" : "waniliowe",\n    "sok" : "pomarańczowy"\n}
```

Chcąc wyświetlić wartość przechowywaną pod konkretnym kluczem działamy podobnie jak z listą.

Różnica polega na tym, że w nawiasach kwadratowych podajemy nie indeks, tylko klucz np.

```
print(menu["zupa"])
```

Jeśli kluczem jest np. jakiś tekst będący nazwą potrawy a wartością lista zawierająca jej składniki to trzymamy się tego przez cały słownik.

Przy konstruowaniu słownika warto być konsekwentnym - nie wrzucamy jako kolejnego elementu np. klucza z jakąś liczbą (niech to będzie chociażby numer strony naszej ulubionej książki kucharskiej) zestawionego z krotką przechowującą kaloryczność składników jako wartość.

Zwracajmy też uwagę na to czy faktycznie w naszym programie korzystamy z tych słownikowych kluczy - jeśli nie, to jest spora szansa, że powinniśmy skorzystać z innej struktury danych.

## Słowniki - dodawanie elementów

Wystarczy przypisać jakąś wartość pod dany klucz - nie trzeba używać żadnej specjalnej metody. Jeśli dany klucz istnieje to wartość przechowywana pod nim zostanie podmieniona, jeśli nie to do słownika zostanie dodana nowa para klucz:wartość.

```
nazwa_zmiennej[klucz] = wartosc
```

np. żeby dodać do słownika menu nowy klucz "przystawki" z jakąś wartością można:

```
menu ={\n    "zupa" : "ogórkowa",\n    "lody" : "waniliowe",\n    "przystawki" : "chleb z masłem"
```

```
"sok" : "pomarańczowy"
```

```
}
```

```
menu["przystawki"] = "sałatka grecka"
```

## Słowniki - usuwanie elementów

Żeby usunąć element ze słownika można skorzystać z metody `.pop()`, podając w nawiasie okrągłym klucz elementu do usunięcia.

Np. chcąc usunąć parę "zupa" : "ogórkowa" z słownika menu wyglądałoby to tak:

```
menu.pop("zupa")
```

## Słowniki - metody

| słownik                | opis                                              |
|------------------------|---------------------------------------------------|
| <code>.keys()</code>   | zwraca listę kluczy słownika                      |
| <code>.values()</code> | zwraca listę wartości słownika                    |
| <code>.pop()</code>    | usuwa element o podanym kluczu                    |
| <code>.items()</code>  | zwraca listę krotek dla każdej pary klucz:wartość |
| <code>.get()</code>    | zwraca wartość dla danego klucza                  |

## Zbiory (set)

Wyglądają podobnie do listy. Mają jednak jedną cenną cechę:

**nie posiadają duplikatów wartości.**

Czyli jeśli dwa razy spróbujemy do zbioru dodać np. słowo "kot" to ten drugi wpis zostanie usunięty.

Nie mają też indeksów (!).

Bezpiecznie jest myśleć o nich jako o takich dziwnych słownikach, które mają jedynie klucze.

Pewnie dlatego też, podobnie jak słowniki, deklaruje się je z użyciem nawiasów klamrowych:

```
moj_zbior={unikat1, unikat2, unikat3}
```

Właśnie ten brak powtórzeń jest często wykorzystywany - np. zamieniamy listę na zbiór żeby pozbyć się duplikatów.

## Zamiana listy na zbiór... i z powrotem :)

```
ksiazki = ["Endymion", "Hobbit", "Fundacja", "Endymion"]
zbior_ksiazek = set(ksiazki)
ksiazki_bez_powtorzen = list(zbior_ksiazek)
```

## Które typy są modyfikowalne (mutable)?

| typ              | modyfikowalny | niemodyfikowalny |
|------------------|---------------|------------------|
| liczba całkow.   |               | tak              |
| liczba zmiennop. |               | tak              |
| wart.log         |               | tak              |
| ciąg znaków      |               | tak              |
| krotka           |               | tak              |
| lista            | tak           |                  |
| słownik          | tak           |                  |
| zbiór            | tak           |                  |

## Odnośniki

„3.12.5 Documentation”. Dostęp 18 sierpień 2024. <https://docs.python.org/3/>.

Viafore, Patrick. *Robust Python: write clean and maintainable code*. First edition. Beijing [China]; Boston [MA]: O'Reilly, 2021.