Wrocław, 1 października 2022

# **P**rogramowanie **w j**ęzyku **P**ython — Laboratorium 2

Podstawowe elementy języka - prymitywne typy danych, kolekcje

prowadzący: K. Kluwak

## Typy sekwencyjne

Typy sekwencyjne pozwalają na zapis w jednej zmiennej wielu wartości. Do najważniejszych typów sekwencyjnych należą:

- listy
- krotki
- łańcuchy znaków
- słowniki
- zbiory

Poszczególne typy sekwencyne różnią się szczegółami technicznymi oraz potencjalnymi zastosowaniami. Do najważniejszych różnic zaliczyć można:

- różne sposoby dostępu do poszczególnych wartości (indeks lub klucz),
- możliwość późniejszych modyfikacji przechowywaia wartości lub dodawania nowych (listy i słowniki są mutowalne, krotki, łańcuchy znaków i zbiory nie),
- typami danych jakie przechowują (możliwość przechowywania danych o różnych lub o identycznych typach).

### Listy

Lista - przechowuje wiele wartości, do których posiadamy dostęp przez tzw. indeks. Przykład: `a = [3, 7, 2, 8, 10]`

Pierwszy element ma indeks 0. Do ostatniego elementu tablicy możemy uzyskać dostęp wprowadzając jako indeks długość listy pomniejszoną o 1 `(len(nazwa_listy)-1)`. Indeksy mogą mieć wartości ujemne - wówczas -1 odnosi się do ostatniego elementu tablicy. Zakres elementów (podzbiór z listy) wyświetlamy korzystając z dwukropka (składnia: `nazwa_listy[0:3]` - instrukcja zwróci pierwsze 3 wartości przechowywane w liście).

Przegląd najważniejszych metod i funkcji działających na listach:

- `append()` - dodawanie elementu do listy (składnia: `nazwa_listy.append(wartosc)`)
- `del` - usuwanie elementu listy (składnia: `del nazwa_listy[indeks]`)
- `insert()` - wstawianie elementu do listy przed wskazany indeks (składnia: `nazwa_listy.insert(przed_który_indeks, wartosc)`)
- `remove()` - usuwanie wskazanej wartości (składnia: `nazwa_listy.remove(wartosc)`)
- `count()` - zliczanie wskazanej wartości (składnia: `nazwa_listy.count(wartosc)`)
- `reverse()` - odwracanie kolejności elementów (składnia: `nazwa_listy.reverse()`)
- `pop()` - wyciąganie, wyświetlanie i usuwanie z listy wskazanego elementu (składnia: `nazwa_listy.pop(i)`)
- `len()` - funkcja zwracająca długość listy (składnia: `len(nazwa_listy)`)
- `sort()` - sortowanie listy (składnia: `nazwa_listy.sort()`)
- `in` - instrukcja umożliwiająca sprawdzenie czy w liście znajduje się wskazana wartość. Zwratą w wyniku logiczną prawdę lub fałsz (składnia: `wartosc in lista`)
- `list()` - funkcja konwertująca wskazaną sekwencję na listę (składnia: `list(sekwencja)`).

Po utworzeniu i zainicjowaniu wartości listy nadal posiadamy możliwość zmiany wprowadzonych wartości oraz dodawania nowych wartości do listy. Dane w liście mogą być różnych typów, np. `a = [1, '2', 3]`

Przydatne instrukcje:

- tworzenie pustej listy: `jakas_zmienna = []`
- wyświetlanie pozbioru listy: `jakas_lista[indeks_poczatkowy:indeks_koncowy]`

Zad 2.1 *Napisz program, który poprosi użytkownika o podanie swojego imienia 5 razy. Program zapisze odpowiedzi do listy, następnie wyświetli komunikat "Cześć imie!!".*

Zad 2.2 *Napisz program, który przyjmie od użytkownika 5 wartości, wprowadzi je do listy, a następnie wyświetli pierwszy i ostatni element. Zaproponuj dwa sposoby wykonania tego zadania.*

Zad 2.3 *Utwórz program, który pobierze od użytkownika 5 wartości, zapisze je do listy, następnie obliczy ich sumę i średnią i wypisze je na ekranie.*

Zad 2.4 *Napisz program, w którym do listy przypisane zostaną 4 zmienne różnych typów, a następnie zostaną wyświetlone na ekranie.*

Zad 2.5 *Rozważ następującą listę:*

```
dane = [1, 1, 3, 2, 4, 6, 5, 3, 2, 1, 6, 50, 2, 3]
```

*Napisz program, który:*

(a) wyświetli utworzoną listę
(b) określi i wyświetli długość listy (liczbę przechowywanych elementów) (funkcja `len()`),
(c) wyświetli 4 pierwsze wartości (operator `:`),
(d) zliczy i wyświetli ile w liście znajduje się wartości 1, 2 i 3 (metoda `count()`),
(e) sprawdzi czy lista zawiera wartość 6 (instrukcja `in`),
(f) wstawi na miejsce po trzecim elemencie wartość -100 (metoda `insert()`) i wyświetli zaktualizowaną listę,
(g) usunie ostatni element w liście (instrukcja `del`) i wyświetli zaktualizowaną listę,
(h) usunie wartość 50 (metoda `remove()`) i wyświetli zaktualizowaną listę,
(i) wyciągnie pierwszy element z listy (metoda `pop()`) i wyświetli zaktualizowaną listę,
(j) utworzy kopię listy dane i odwróci ich kolejność (metoda `reverse()`) i wyświetli otrzymaną listę,
(k) utworzy kopię listy dane i posortuje jej elementy (metoda `sort()`) i wyświetli otrzymaną listę.

**Krotki**

Krotki (tuples) to kolejny typ sekwencyjny - podobny do listy, posiadający jednak pewne istotne różnice:

- krotkę tworzymy korzystając z nawiasów okrągłych (a nie kwadratowych jak w przypadku list),
- zasadniczo wartości przechowywanych w krotkach nie można zmieniać. Nie można również dodawać elementów do ktorek,
- krotki służą do przechowywania niezmiennych elementów
- Krotki mogą przydatne wówczas, gdy chcemy dzielić z innymi użytkownikami (lub procesami) dane, ale nie chcemy zezwolić na ich modyfikację.

Mechanizm ochrony danych w krotkach można jednak obejść poprzez nadpisywanie na dotychczasowej krotce nowej krotki. Do elementów krotki mamy dostęp (tak jak w przypadku list) za pośrednictwem indeksu. Krotki mogą przechowywać dane różnych typów.

Przegląd najważniejszych metod i funkcji działających na krotkach:

- count() - zlicza wystąpienia wskazanej wartości w krotce
- index() - zwraca indeks pierwszej wskazanej wartości w krotce

Zad 2.6 *Napisz program, w którym utworzysz krotkę, przechowującą zestaw pięciu czteroznakowych kluczy dostępu. Wyświetl pierwszy i ostatni element. Spróbuj podmienić lub usunąć wartość dowolnego elementu w krotce (korzystając np. z instrukcji* `krotka[indeks] = inna_wartosc oraz z del krotka[indeks]`).

Zad 2.7 *Napisz program, w którym utworzysz krotkę:*

```
dane = (1, 1, 3, 2, 4, 6, 5, 3, 2, 1, 6, 50, 2, 3)
```

Sprawdź:

(a) jaka jest długość krotki,

(b) ile razy występują w niej wartości 1, 2, 3,

(c) czy krotka zawiera wartość 50.

(d) Zaproponuj sposób wstawienia do krotki dane po czwartym elemencie (liczba 2) kolejnych wartości 100, 200, 300.

**Typ napisowy (String)**

Napis (string) to sekwencja znaków alfanumerycznych. W odróżnieniu od poznanych wcześniej list i krotek - wszystkie elementy napisu mają ten sam typ - str. Do kolejnych elementów sekwencji mamy dostęp za pośrednictwem indeksu. Zasadniczo typ napisowy przechowuje teksty, a w języku Python mamy bogaty zasób metod edytujących tekst. Poszczególne elementy sekwencji jak i sama sekwencja mogą być zmieniane. Typ napisowy inicjalizujemy korzystając z cudzysłowu. Przykład: `napis = "Ala ma kota"`

Przegląd najważniejszych metod:

- `capitalize()` - zmiana pierwszej litery sekwencji na wielką (składnia: `napis.capitalize()`),
- `islower()` - sprawdzenie, czy wszystkie znaki w napisie to małe litery (składnia: `napis.islower()`),
- `isupper()` - sprawdzenie, czy wszystkie znaki w napisie to wielkie litery (składnia: `napis.isupper()`),
- `count(podciag)` - zliczanie wystąpień podciągu znaków w pierwotnym napisie (składnia `napis.count()`),
- `isdigit()` - sprawdzenie, czy znaki w napisie są liczbami (składnia: `napis.isdigit()`),
- `replace(old_str, new_str)` - podmiana starej sekwencji nową (składnia: `napis.replace(stara_sek, nowa_sek)`),
- `strip()` - usuwanie początkowych i końcowych białych znaków,
- `len()` - funkcja zwracająca długość (liczbę znaków) w napisie (składnia: `len(napis)`).

Zad 2.8 *Napisz program, którzy przyjmie od użytkownika dowolne zdanie. Następnie program wyświetli ile znaków zawiera wprowadzony tekst oraz ile znajduje się w nim spacji.*

Zad 2.9 *Napisz program, który pobierze od użytkownika dowolny tekst, a następnie:*

*(a) wyświetli informację, czy wszystkie litery w tekście są wielkie,*

*(b) wyświetli informację, czy wszystkie litery w tekście są małe,*

*(c) zastąpi pierwsze 3 znaki słowem "ABC".*

Zad 2.10 *Napisz program, w którym do zmiennej napis przypiszesz tekst "Dzisiaj znowu poznajemy Pythona!". Następnie:*

*(a) zamień słowo "Pythona" na "interesujące techniki programowania",*

*(b) zlicz wystąpienia litery "i",*

*(c) wyświetl długość nowego tekstu.*

**Słowniki**

Słownik (dictionary) to struktura danych zawierająca przypisane do siebie pary klucz : wartość. W obrębie słownika do przechowywanych elementów mamy dostęp nie za pośrednictwem indeksu, ale za pośrednictwem tzw. klucza. Klucz w słowniku nie może się powtarzać. Elementy w słowniku nie są uporządkowane (nie są ułożone po kolei tak jak to było w listach, krotkach i napisach). Innymi słowy - kolejność nie ma znaczenia. Elementy słownika mogą mieć różne typy. Dotyczy to rówież klucza (wszystko może być kluczem bądź wartością). Słowniki możemy edytować - dodawać do nich elementy oraz je modyfikować. Słowniki inicjalizujemy korzystając ze znaku klamer , pary kluczy i wartości są łączone drukropkiem, kolejne elementy słownika rozdzielane przecinkiem. Przykład:

```
slownik = {klucz1 : wartosc1, klucz2 : wartosc2, klucz3 : wartosc3}
```

Kolejne elementy do słownika możemy prowadzać odnosząc się do nazwy słownika, w nawiasie kwadratowym wprowadzić nowy klucz a po operatorze przypisania podać wartość. Przykład:

```
slownik[wartosc4] = klucz4
```

Przegląd najważniejszych metod działających na słownikach:

- `clear()` - usuwa wszystkie elementy ze słownika (składnia: `slownik.clear()`),
- `copy()` - zwraca kopię słownika (składnia `slownik.copy()`),
- `fromkeys(keys, values)` - zwraca słownik na postawie podanych zbiorów kluczy i wartości (składnia: `slownik.fromkeys(klucze, wartosci)`),
- `get()` - zwraca wartość dla danego klucza (składnia: `slownik.get()`),
- `items()` - zwraca listę krotek wszystkich par kluczy i wartości (składnia: `slownik.items()`),
- `pop(key)` - wyciąga i usuwa wskazaną wartość (na podstawie klucza) (skladnia: `slownik.pop(klucz)`),
- `popitem()` - wyciąga i usuwa ostatnią wprowadzoną parę klucz-wartość (składnia: `slownik.popitem()`),
- `setdefault(key)` - zwraca wartość dla danego klucza. Jeżeli klucz nie istnieje - wstawia klucz (ten, który został wywołany) (składnia: `slownik.setdefault(klucz)`),
- `update(key: value)` - pozwala na dodawanie do słownika kolejnych par klucz-wartość (składnia: `slownik.update({klucz: wartość)}`),
- `values()` - zwraca wszystkie wartości ze słownika (składnia: `słownik.values()`).

Zad 2.11 *Utwórz program, który wykorzystując słownik, będzie pobierać od użytkownika numer miesiąca, a wyświetlać na ekranie będzie jego nazwę w formie tekstowej.*

Zad 2.12 *Utwórz program, który wykorzystując słownik, będzie tłumaczyć liczby 0-10 na słowa.*

Zad 2.13 *Korzystając z informacji zawartych na stronie: https://pl.wikipedia.org/wiki/Rzymski_system_zapisywania_liczb utwórz program, który zamieni cyfrę rzymską na arabską (do 1000).*

Zad 2.14 *Napisz program, który wprowadzoną przez użytkownika liczbę 5-cyfrową wyświetli w formie słownej (np. 12345 - jeden dwa trzy cztery pięć).*

Zad 2.15 *Napisz program, w którym zamieścisz następujący słownik:*

```
slownik = {"m1": "Ford", "m2": "Fiat", "m3": "Opel", "m4": "Skoda", "m5": "Wołga", "m6": "Dacia"}
```

*następnie:*

*(a) wykonaj kopię słownika (metoda copy()),*

*(b) zwróć listę krotek wszystkich par kluczy i wartości (metoda items()),*

*(c) dodaj parę "m7": "FSO" (metoda update()),*

*(d) wykonaj kolejną kopię słownika i wyczyść jego zawartość (metody copy() i clear()).*

**Zbiory**

Zbiór to kolejny typ sekwencyjny. Posiada ważną własność - jego elementy nie mogą się powtarzać. Podobnie jak słowniki - są nieuporządkowane (kolejność elementów jest bez znaczenia). Zbiory są edytowalne - możemy dodawać bądź usuwać elementy ze zbioru. Zbiory inicjalizujemy z wykorzystaniem funkcji set() lub wprowadzając zestaw danych otoczonych klamrami . Kolejne elementy zbioru są rozdzielone przecinkiem. Mogą przechowywać elementy dowolnych typów. Nie mamy możliwości wywoływania określonych elementów - zbiory nie posiadają ani indeksów ani kluczy. Możemy za to wykonywać działania na zbiorach. Przykład

```
a = {1, 2, 3, 2.1, 2, 3, 4, "4"}
```

Przegląd najważniejszych metod i funkcji działających na zbiorach:

- `add()` - dodaje element do zbioru (składnia: `zbior.add(element)`),
- `clear()` - usuwa wszystkie elementy ze zbioru (składnia: `zbior.clear()`),
- `copy()` - zwraca kopię zbioru (składnia: `zbior.copy()`),

- `difference()` - zwraca różnicę zbiorów (składnia: `zbior1.difference(zbior2)`),
- `difference_update()` - usuwa elementy ze zbioru, które są obecne w drugim zbiorze (składnia: `zbior1.difference_update(zbi`
- `discard()` - usuwa wskazany element (składnia: `zbior.discard(element)`)
- `intersection()` - zwraca część wspólną zbiorów (składnia: `zbior1.intersection(zbior2)`),
- `intersection_update()` - usuwa ze zbioru elementy nieobecne w drugim (składnia: `zbior1.intersection_update(zbior2)`),
- `isdisjoint()` - zwraca informację czy zbiory posiadają część wspólną czy nie (składnia: `zbior1.isdisjoint(zbior2)`),
- `issubset()` - zwraca informację czy zbiór jest podzbiorem drugiego (składnia: `zbior1.issubset(zbior2)`),
- `issuperset()` - zwraca informację czy zbiór zawiera się w drugim (składnia: `zbior1.issuperset(zbior2)`),
- `pop()` - zwraca i usuwa pewien element ze zbioru (składnia: `zbior.pop()`),
- `remove()` - usuwa wskazany element ze zbioru (składnia: `zbior.remove(element)`),
- `symmetric_difference()` - zwraca różnicę symetryczną zbiorów (składnia: `zbior1.symmetric_difference(zbior2)`),
- `symmetric_difference_update()` - usuwa elementy obecne w obu zbiorach oraz wstawia elementy, które nie są obecne w obu zbiorach (składnia: `zbior1.symmetric_difference_update(zbior2)`),
- `union()` - zwraca zbiór zawierający wszystkie elementy z pierwszego i drugiego zbioru(składnia: `zbior1.union(zbior2)`),
- `update()` - wstawia elementy jednego zbioru do drugiego zbioru (składnia: `zbior1.update(zbior2)`).

Zad 2.16 *Napisz program, w którym użytkownik poda po 4 elementy (liczby całkowite) do dwóch różnych zbiorów. Następnie program wyświetli:*

*(a) sumę zbiorów*

*(b) różnicę zbiorów*

*(c) część wspólną zbiorów.*

### Konwersja typów sekwencyjnych

Aby dokonać konwersji jednego typu sekwencyjnego na drugi korzystamj z instrukcji:

- konwersja na listę: `list(krotka albo zbiór)`
- konwersja na zbiór: `set(krotka albo lista)`
- konwersja na krotkę: `tuple(lista albo zbiór)`