

Kolekcje (struktury danych) w Pythonie

- listy (`list`)
- krotki (`tuple`)
- słowniki (`dict`)
- zbiory (`set`)
- pokrewne: napisy (`string`)

Lista

Lista to **zbiór elementów**, np. liczb lub napisów. W innych językach często zwana tablicą.

Listy mają określoną **długość** i **kolejność** elementów. Elementy mogą się powtarzać. Nie muszą być uszeregowane kolejno wartościami.

Pozycja elementów w liście to **indeks**. W Pythonie indeksy numeruje się od 0 !

Przykład listy: **tab = [5, 15, 1, 25, 2]**

Listy – operacje na elementach

tab = [5, 15, 1, 25, 2]

tab[0] -> 5

tab[2] -> 1

tab[-1] -> 2

tab[-3] -> 1

tab[10] -> błąd

Listy - własności

```
tab = [5, 15, 1, 25, 2]
```

```
len(tab) -> 5
```

```
min(tab) -> 1
```

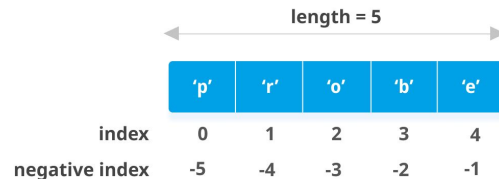
```
max(tab) -> 25
```

```
sum(tab) -> 48
```

```
sum(tab)/len(tab) -> 9.6
```

```
sorted(tab) -> [1, 2, 5, 15, 25]
```

Listy w Pythonie - co o nich wiemy ?



A diagram illustrating a Python list. A horizontal double-headed arrow at the top is labeled "length = 5". Below it, a row of five blue boxes contains the characters 'p', 'r', 'o', 'b', and 'e'. Underneath these boxes, two rows of indices are shown. The first row, labeled "index", contains the values 0, 1, 2, 3, and 4. The second row, labeled "negative index", contains the values -5, -4, -3, -2, and -1.

	'p'	'r'	'o'	'b'	'e'
index	0	1	2	3	4
negative index	-5	-4	-3	-2	-1

- Każda lista ma określoną długość
- Każdy element ma swoją pozycję (indeks), zaczynamy je liczyć od 0 !
- W Pythonie możemy też liczyć elementy od końca (-1)
- możemy policzyć takie wartości jak minimum, maksimum, suma, średnia (jeśli elementami są TYLKO liczby!)
- listy można posortować wartościami (jeśli elementami są TYLKO liczby lub TYLKO napisy)

Ćwiczenie: Stwórz listę z imionami 4 osób w grupie (lista NAPISÓW - o czym musisz pamiętać?). Wyświetl posortowaną alfabetycznie listę.

Operacje na listach

```
my_list = [11,12,13]
```

- modyfikacja elementu listy: `my_list[1] = 15` -> `my_list: [11,15,13]`
- dołączenie elementu do końca:
`my_list.append(18)` -> `my_list: [11,15,13,18]`
- wklejenie elementu do listy:
`my_list.insert(2, 14)` -> `my_list: [11,15,14,13,18]`
- "zdjęcie" elementu z końca i zwrócenie:
`x = my_list.pop()` -> `my_list: [11,15,14,13], x: 18`
- odwrócenie listy: `my_list.reverse()` -> `my_list = [13,14,15,11]`

Operacje na listach

```
my_list1 = [1,2,3] i my_list2 = [4,5]
```

- łączenie list:

```
large_list = my_list1 + my_list2 -> large_list: [1,2,3,4,5]
```

- mnożenie list przez liczbę:

```
new_list = large_list * 3 -> new_list:[1,2,3,4,5,1,2,3,4,5,1,2,3,4,5]
```

- wycinek listy:

- `new_list[0:4]` lub `new_list[:4]` -> `[1,2,3,4]` - odliczanie od 0 można pominąć
- `new_list[8:15]` lub `new_list[8:]` lub (!) `new_list[8:1000]` -> `[4,5,1,2,3,4,5]`
 - odliczanie do końca można pominąć
- `new_list[1:8:2]` -> `[2,4,1,3]` - domyślny krok +1 można pominąć
- `new_list[4:0:-1]` -> `[5,4,3,2]` - można odliczać z ujemnym krokiem
- `new_list[::-1]` -> `[5,4,3,2,1,5,4,3,2,1,5,4,3,2,1]` - odwrócenie listy

Lista różnych typów i lista 2D

W Pythonie ze względu na dynamiczne typowanie możliwe jest utworzenie listy z elementami różnego typu, np.

```
x = [1, 5.0, "mama", "tata", [1,2,3], 99]
```

Lista dwuwymiarowa – lista składająca się z innych list

```
x = [[1,2,3],[8,9,10]]
```

```
x[0] -> [1,2,3]
```

```
x[0][1] -> 2
```

```
x[1][-1] -> 10
```


Krotki (tuple)

Krotki - "to takie prawie listy, ale niemodyfikowalne"

Tworzenie krotki:

- składnia: `nazwa_zmiennej = (element1, element2, element3...)` - uwaga: nawiasy zwykłe!
- przykład: `my_tuple = (22,55,66)`
- inny dozwolony zapis: `my_tuple = 22, 55, 66` (brak nawiasów, niezalecane)

Własności szczególne krotki `my_tuple`:

- krotki są niemodyfikowalne -> nie można dodać elementu, usunąć, rozszerzyć, skrócić
- mają ustaloną długość i zawartość
- można je do siebie dodać lub pomnożyć (ale wynikiem będzie nowa krotka):

`x = (1,2) + (3,4) -> x: (1,2,3,4)`

Po co?

- zabezpieczenie przed modyfikacją czegoś, czego nie można modyfikować
- lepsza optymalizacja niż dla list

Słownik (dict)

Słownik - zawiera pary klucz - wartość; klucze muszą być unikalne (nie mogą się powtarzać)

Tworzenie słownika:

- składnia: `nazwa_zmiennej = { klucz1 : wartość1, klucz2 : wartość2, ... }`
- uwaga: nawiasy klamrowe i dwukropek
- przykład: `my_dict = { "apples" : 5, "bananas" : 10, "oranges": 15 }`

Odwołanie do elementu słownika - analogicznie jak w listach (indeks -> klucz):

```
print( my_dict["apples"] )    -> 5
print( my_dict["bananas"] )   -> 10
print( my_dict["cherries"] )  -> błąd
print( my_dict[1] )           -> błąd
```

Słownik (dict)

Klucz - musi być unikalny, klucze nie mogą się powtarzać i muszą być niemodyfikowalne.

Kluczem może być np. **liczba**, **napis**, **wartość zmiennej**, **krotka (!)**. Nie może to być lista lub inny słownik - wynika to ze sposobu zapisu zmiennej w pamięci fizycznej (typy danych modyfikowalne).

my_dict.keys() - lista kluczy,

my_dict.values() - lista wartości,

my_dict.items() - pary krotek (klucz, wartość)

Słowniki (dict)

Operacje na słownikach: `my_dict = { 1 : "jeden", 2 : "dwa" }`

- dodanie elementu:

`my_dict.update({3 : "trzy"})` -> `{ 1 : "jeden", 2 : "dwa", 3 : "trzy" }`

- dodanie elementu - inaczej:

`my_dict[4] = "cztery"` -> `{ 1 : "jeden", 2 : "dwa", 3 : "trzy", 4 : "cztery" }`

- usunięcie elementu:

`del my_dict[4]` -> `{ 1 : "jeden", 2 : "dwa", 3 : "trzy" }`

Ćwiczenie: stwórz listę zakupów jako słownik (artykuł : kwota). Wyświetl elementy słownika oraz oblicz sumę kwoty wydanej na zakupy.

Zbiory (set)

Zbiór - nieuporządkowane elementy bez powtórzeń

Tworzenie zbioru:

- składnia: `nazwa_zmiennej = { element1, element2, element3 }` - uwaga: nawiasy klamrowe
- przykład: `my_set = {1, 2, 4}`

Operacje na zbiorach:

- dodanie elementu: `my_set.add(5)` -> `my_set: {1,5,2,4}`
- usunięcie elementu: `my_set.remove(4)` -> `my_set: {2,1,5}`
- pobranie i zwrócenie "ostatniego" (losowego) elementu: `x = my_set.pop()` -> `np. my_set={2,5}, x=1`
- sprawdzenie czy element jest w zbiorze: `print(5 in my_set)` -> `True`

Po co?

- lepiej zoptymalizowany niż lista, jeśli nie potrzebujemy znać kolejności
- kiedy chcemy zapewnić niepowtarzalność elementów

Napisy (string)

String - "ciąg znaków" - pewne operacje wykonuje się analogicznie jak w listach/tuplach, napisy są niemodyfikowalne (każda modyfikacja tworzy w pamięci nowy string)

np. `name = "HardCoder"`

- indeksowanie napisów: `print(name[1])` -> "a"
- podzbiory (substring): `print(name[1:4])` -> "ard"
- pozycja elementu: `print (name.find("rd"))` -> 2
- pozycja elementu nieistniejącego: `print (name.find("X"))` -> -1
- zamiana znaków: `print(name.replace("Coder", "Work"))` -> "HardWork"
- zmiana wielkości liter: `print(name.upper() , name.lower())` -> "HARDCODER", "hardcoder"
- podział względem znaku: `print("ty,ja,on".split(","))` -> ["ty", "ja", "on"]
- łączenie napisów z listy pustym znakiem: `print("".join(["a", "b", "c"]))` -> "abc"

Kolekcje wbudowane - charakterystyka

- **listy** (`list`) - zbiory danych o określonej długości, kolejności, modyfikowalne
- **krotki** (`tuple`) - zbiory danych o określonej długości, kolejności, niemodyfikowalne
- **słowniki** (`dict`) - zbiory par klucz-wartość, o określonej wielkości, modyfikowalne (UWAGA! przed Pythonem 3.8 słowniki miały pary w losowej kolejności!)
- **zbiory** (`set`) - zbiory danych niepowtarzalnych w losowej kolejności
- **napisy** (`string`) - zachowujące się jak lista, niemodyfikowalne (technicznie)

Deklaracja kolekcji pustych

- lista:

`x = []` lub `x = list()`

- krotka (uwaga! nie będzie można potem zmodyfikować)

`x = ()` lub `x = tuple()`

- słownik:

`x = {}` lub `x = dict()`

- set:

`x = set()`

Konwersja między typami

- string->list:
`list("alfabet") -> ['a', 'l', 'f', 'a', 'b', 'e', 't']`
- list->string (dodaje znaki nawiasów i spacje po przecinkach):
`"napis"+str([1,2,3]) -> "napis[1, 2, 3]"`
- set->list:
`list({1,5,5,6,10,1}) -> [1,10,6,5]` (kolejność jest losowa)
- string->set:
`set("alfabet") -> {'b', 't', 'f', 'e', 'l', 'a'}` (kolejność losowa)
- list->set:
`set([1,2,1,1,2,2,3]) -> {1,3,2}` (kolejność losowa)

krotki - analogicznie jak listy -> tuple(<zawartość innego typu>)

dict - konwersja bardziej skomplikowana (istnieją dedykowane funkcje)

Ćwiczenia interaktywne - do samodzielnego potestowania wiedzy

1) Własności listy:

<https://learningapps.org/watch?v=ppywyxbok20>

2) Własności kolekcji:

<https://learningapps.org/watch?v=ppvhr2zgc20>

3) Quiz: Co to za kolekcja:

<https://forms.gle/jQADG3637BGRa2VJ9>

Ćwiczenia #2

Za ćwiczenia razem do zdobycia 10 pkt.

Uwaga: wszystkie ćwiczenia zasadniczo da się zrobić bez znajomości pętli i tworzenia własnych funkcji, wykorzystując tylko wiedzę zdobytą dotychczas, czyli operacje na zmiennych, stringach, strukturach danych, if-elif-else. Tylko w zadaniu 4 przydaje się użycie pętli for lub jednolinijkowca list comprehension do zamiany elementów na integer. Śmiało korzystaj z dokumentacji i stron z sieci, do wyszukania odpowiednich brakujących informacji i funkcji nieprzedstawionych wprost na zajęciach. Warianty "dla zaawansowanych" albo "możesz wyszukać w sieci" są przeznaczone jako większe wyzwania dla osób znających nieco lepiej Pythona i nie są uwzględniane w punktacji.

- [2pkt] Rozszerzenie zadania 5 z poprzedniego zestawu - pola figur

Napisz prosty program do obliczania pola figur geometrycznych. Użytkownik wybiera pole figury spośród dostępnych w zdefiniowanej liście 3-elementowej (np. prostokąt, trapez, koło, trójkąt, deltoid) - wpisując odpowiedniego stringa. Sprawdź, czy wpisana nazwa znajduje się w liście, jeśli nie, program kończy się. Następnie dla wybranej figury użytkownik podaje parametry do obliczenia pola w odpowiedniej ilości (bok, wysokość, średnica itp.), w ramach tylko jednej instrukcji input(), oddzielone spacją. Następnie zapisz parametry dla każdej figury w jednej liście, wykorzystaj split(). Zweryfikuj, czy liczba parametrów jest poprawna i czy są to wartości dodatnie, jeśli nie, program kończy się. Następnie oblicz i wyświetl pole danej figury, ale do obliczeń wykorzystując tylko elementy listy i odpowiednie operacje indeksowania, bez pomocniczych zmiennych do parametrów figury.
- [2pkt] Zdefiniuj słownik, którego wartościami będą wydatki na życie w ostatnich kilku miesiącach, a kluczami nazwy miesięcy. Wyznacz i wyświetl wartość minimalną, maksymalną, sumę i wartość średnią (wykonaj odpowiednie operacje na liście wartości). Sprawdź czy kwota za ostatni miesiąc (nie istnieje prosty sposób na "automatyczne" wybranie ostatniego miesiąca ze słownika, odwołaj się po prostu do nazwy) przekracza wartość średnią - jeśli tak, to wyświetl tekst ostrzeżenia "zaczynj oszczędzać", a jeśli nie, informację "jesteś bezpieczny".
- [2pkt] Formularz rejestracji

Poproś użytkownika o wpisanie (inputy) imienia, nazwiska, daty urodzenia w narzuconym formacie dd-mm-rrr oraz maila. Przy pomocy operacji na elementach listy, napisz proste walidatory dla daty urodzenia i maila (ma spełniać tylko prosty schemat "tekst@tekst", zawierać co najmniej jedną kropkę i nie zaczynać się od cyfry, bez konieczności użycia wyrażeń regularnych, poszukaj odpowiednich funkcji dla stringów do sprawdzania czy znak jest literą, liczbą, itd). Następnie przedstaw wpisane dane w postaci słownika, zawierającego zaszyfrowane dane w schemacie: {imię i nazwisko: P**** M*** (złączone razem dane z dwóch inputów, połączone spacją, tylko pierwsze litery, zamienione na duże jeśli użytkownik wpisał małe, reszta w postaci gwiazdek); wiek: 32 (obliczony na podstawie tylko roku urodzenia); mail: p*****@gmail.com (tylko pierwsza litera maila, reszta przed @ to gwiazdki, domena zostaje bez zmian)}
- [4pkt] Odczyt danych i walidacja numeru PESEL

Użytkownik wpisuje numer PESEL w postaci stringa (zahardkodowana wartość na początku kodu lub input). Przedstaw go w postaci listy lub krotki (każda cyfra pod osobnym indeksem). Uwaga: cyfry numeru PESEL po podzieleniu na znaki będą wciąż stringami, dlatego może się przydać operacja rzutowania wszystkich elementów na liczby - niestety wymaga to zastosowania pętli for lub list comprehension - wykorzystaj jeden ze sposobów z [Python I Converting all strings in list to integers - GeeksforGeeks](#). Można też tego uniknąć i rzutować za każdym razem elementy do typu int przy obliczeniach (mało wygodne i zgrabne, ale możliwe). Wykorzystując wiadomości z [PESEL - Wikipedia, wolna encyklopedia](#) oraz operacje na elementach listy, zwaliduj poprawność wpisanego numeru PESEL (długość, dzień, miesiąc i rok, cyfra kontrolna - tu stwórz pomocniczą listę z wagami; ogranicz się tylko do urodzonych w stuleciach 1900+ i 2000+) - w przypadku braku spełnienia któregoś kryterium wyświetl stosowną informację; a jeśli PESEL jest poprawny, odczytaj z niego i przedstaw w formie słownej: datę urodzenia* oraz płeć (* - miesiąc może być podany liczbowo, ale możesz też wyszukać w sieci prosty sposób na pozyskanie nazwy miesiąca w Pythonie). Do testowania skorzystaj z generatora online, np. <http://generatory.it/> lub (dla zaawansowanych) zapoznaj się z możliwościami biblioteki Faker ([Locale pl_PL — Faker 24.0.0 documentation](#))