Kolekcje w Pythonie

M@я3k Pų𝐝€£kØ

Programowanie w Pythonie

Spis treści

- Kolekcja
- Właściwości kolekcji
- Tworzenie kolekcji
- Wyświetlanie kolekcji
- Nieduplikowalność kolekcji
- Dodawanie elementu do kolekcji
- Usuwanie elementu z kolekcji
- Opróżnianie kolekcji
- Kopiowanie kolekcji
- Liczba elementów kolekcji
- Sprawdzenie przynależności do kolekcji
- Przeglądanie zawartości kolekcji
- Modyfikacja zawartości kolekcji
- Operacje na zbiorach

Kolekcja

- Kolekcje (set) są zdefiniowane jako zbiory niezmiennych, nieuporządkowanych obiektów.
  - W przeciwieństwie do list, które mogą zawierać duplikaty, kolekcje przechowują tylko unikalne elementy.
- Kolekcje są implementacją matematycznej koncepcji zbioru, gdzie kolejność elementów nie ma znaczenia, a duplikaty są ignorowane.

Właściwości kolekcji

- **Usuwanie duplikatów**:
- Kolekcje są idealne do usuwania duplikatów z list lub innych iterowalnych obiektów.
- Funkcja set(), która jest bardziej wydajna niż użycie pętli.
- **Sprawdzanie przynależności**:
- Operacja in pozwala szybko sprawdzić, czy dany element znajduje się w kolekcji.
- **Operacje na zbiorach**:
- Kolekcje obsługują operacje logiczne takie jak suma, różnica, iloczyn i różnica symetryczna.
- **Analiza danych**:
- Kolekcje mogą być używane do analizy danych, np. do znajdowania unikalnych wartości w kolumnie tabeli.
- **Uczenie maszynowe**:
- Kolekcje są używane w algorytmach uczenia maszynowego, np. do reprezentowania zbiorów cech lub klasyfikacji.
- **Automatyzacja zadań**:
- Kolekcje mogą być używane do tworzenia unikalnych identyfikatorów, np. dla plików lub elementów interfejsu użytkownika.

Tworzenie kolekcji

- Kolekcja pusta

```
kolekcja = set ()
print (kolekcja)          #set()
```

- Podanie wartości w momencie tworzenia (w klamrach { })

```
kolekcja = set ({1, 2, 3, 4, 5})
print (kolekcja)          #{1, 2, 3, 4, 5}
```

- Zrobienie kolekcji z listy

```
lista = (1, 1, 2, 2, 3, 4, 5, 6, 6)
kolekcja = set (lista)
print (kolekcja)          #{1, 2, 3, 4, 5, 6}
```

Wyświetlanie kolekcji

- Kolekcja (**set**) jest zawsze wyświetlana w klamrach {}

```
kolekcja = set ({1, 2, 3, 4, 5})
print (kolekcja) #{1, 2, 3, 4, 5}
```

- Wyświetlenie typu informuje, ze jest to kolekcja (**set**)

```
print (type(kolekcja))
                    #<class 'set'>
```

Nieduplikowalność kolekcji

- Kolekcja (**set**) jest zawsze zbiorem unikalnych elementów. W dany zbiorze może znaleźć się tylko jeden element o danej wartości.
- Zduplikowane elementy są usuwane.

```
kolekcja = set ({1,1,2,2,3,3,4,4,5,5})
print (kolekcja)#{1, 2, 3, 4, 5}
```

- Próba dodania zduplikowanego elementu kończy się brakiem efektu

```
kolekcja.add (5)
print (kolekcja)     #{1, 2, 3, 4, 5}
kolekcja.add (6)
print (kolekcja)     #{1, 2, 3, 4, 5, 6}
```

Dodawanie elementu do kolekcji

- Dodawanie elementu do zbioru za pomocą metody wewnętrznej **add**.
  - Jeśli element będzie zduplikowany, nie zostanie dodany.

```
kolekcja = set ({1, 2, 3, 4, 5})

kolekcja.add (5)
print (kolekcja)      #{1, 2, 3, 4, 5}
kolekcja.add (6)
print (kolekcja)      #{1, 2, 3, 4, 5, 6}
```

Usuwanie elementu z kolekcji

- Za pomocą metody wewnętrznej remove:

```
kolekcja = set ({1, 2, 3, 4, 5})
kolekcja.remove (5)
print (kolekcja)          #{1,2,3,4}
```

- Gdy dany element nie istnieje w zbiorze generowany jest kod błędu.

```
kolekcja.remove (6)

Traceback (most recent call last):
  File "/home/main.py", line 15, in
   <module>
    kolekcja.remove (6)
    ~~~~~~~~~~~~~~~~^^^
KeyError: 6
```

- Gdy nie ma pewności, czy dany element istnieje w zbiorze lepiej użyć metody **discard**, która usuwa dany element, jeśli istnieje on w zbiorze:

```
kolekcja.discard(6)
print (kolekcja)          #{1,2, 3, 4}
```

- Metoda, która usuwa i zwraca pierwszy element zbioru jest pop:

```
kolekcja = set ({1, 2, 3, 4, 5})
print (kolekcja)#{1, 2, 3, 4, 5}
print(kolekcja.pop())    #1
print(kolekcja)          #{2, 3, 4, 5}
```

- Metoda pop zwraca błąd, jeżeli zbiór jest pusty.

```
kolekcja = set ()
print(kolekcja.pop())

Traceback (most recent call last):
  File "/home/main.py", line 27, in
   <module>
    print(kolekcja.pop())        #1
          ~~~~~~~~~~~~^^
KeyError: 'pop from an empty set'
```

Opróżnianie kolekcji

- Opróżnianie kolekcję za pomocą metody **clear**:

```
kolekcja = set ({1, 2, 3, 4, 5})
print (kolekcja)  #{1, 2, 3, 4, 5}
kolekcja.clear()
print(kolekcja)       #set()
```

Kopiowanie kolekcji

- Kopiowanie zbioru za pomocą metody copy:

```
kolekcja1 = set ({1, 2, 3, 4, 5})
kolekcja2 = kolekcja1.copy()
print (kolekcja1) #{1, 2, 3, 4, 5}
print (kolekcja2) #{1, 2, 3, 4, 5}
```

Liczba elementów kolekcji

- Za pomocą funkcji **len**:

```
kolekcja = set ({1, 2, 3, 4, 5})
print (len (kolekcja))          #5
```

- Za pomocą metody wewnętrznej **__len__**:

```
kolekcja = set ({1, 2, 3, 4, 5})
print (kolekcja.__len__())          #5
```

Sprawdzenie przynależności do kolekcji

- Za pomocą polecenia **in**:

```
kolekcja = set ({1, 2, 3, 4, 5})
print(2 in kolekcja)  #True
print(6 in kolekcja)  #False
```

Przeglądanie zawartości kolekcji

- Kolekcje są nieuporządkowane, więc nie można uzyskać dostępu do elementu za pomocą indeksu, jak w przypadku list.

```
kolekcja = set ({1, 2, 3, 4, 5})
for i in range (len (kolekcja)):
    print (kolekcja[i])
```

```
Traceback (most recent call last):
  File "/home/main.py", line 35, in <module>
    print (kolekcja[i])
          ~~~~~~~~~^^^
TypeError: 'set' object is not subscriptable
```

- Aby uzyskać dostęp do elementów kolekcji, można iterować po nim lub użyć funkcji in.

```
kolekcja = set ({1, 2, 3, 4, 5})
for i in kolekcja:
    print (i)
```

Modyfikacja zawartości kolekcji

- Kolekcje zawierają tylko niezmienne obiekty. Próba modyfikacji elementu zbioru spowoduje błąd.

```
kolekcja = set ({1, 2, 3, 4, 5})
kolekcja[i] = 0

Traceback (most recent call last):
  File "/home/main.py", line 41, in <module>
    kolekcja[i] = 0
    ~~~~~~~~~^^^
TypeError: 'set' object does not support item assignment
```

- Aby zmodyfikować element zbioru, należy usunąć go i dodać z powrotem zmodyfikowaną wersję.

```
kolekcja = set ({1, 2, 3, 4, 5})
kolekcja.pop ()
print (kolekcja)            #{2, 3, 4, 5}
kolekcja.add (0)
print (kolekcja)            #{0, 2, 3, 4, 5}
```

Operacje na zbiorach cz.1

- Łączenie dwóch zbiorów
  - Za pomocą wewnętrznej metody union:

```
zbior1 = set("tekst")
zbior2 = set("tekst jakiś tam")
print(zbior1.union(zbior2))
```

  - Za pomocą operatora |:

```
zbior1 = set("tekst")
zbior2 = set("tekst jakiś tam")
print(zbior1 | zbior2)
```

- Część wspólna dwóch zbiorów
  - Za pomocą wewnętrznej metody intersection:

```
zbior1 = set("tekst")
zbior2 = set("tekst jakiś tam")
print(zbior1.intersection(zbior2))
```

  - Za pomocą operatora &

```
zbior1 = set("tekst")
zbior2 = set("tekst jakiś tam")
print(zbior1 & zbior2)
```

Operacje na zbiorach cz.2

- Różnica dwóch zbiorów
  - Za pomocą wewnętrznej metody difference:

```
zbior1 = set("tekst")
zbior2 = set("tekst jakiś tam")
print(zbior1.difference(zbior2))
print(zbior2.difference(zbior1))
```

  - Za pomocą operatora -:

```
zbior1 = set("tekst")
zbior2 = set("tekst jakiś tam")
print(zbior1 - zbior2)
print(zbior2 - zbior1)
```

- Różnica symetryczna dwóch zbiorów (Dany element występuje w jednym lub w drugim zbierze, ale nie w obu na raz)
  - Za pomocą wewnętrznej metody symmetric_difference:

```
zbior1 = set("tekst")
zbior2 = set("tekst jakiś tam")
print(zbior1.symmetric_difference(zbior2))
```

  - Za pomocą operatora ^:

```
zbior1 = set("tekst")
zbior2 = set("tekst jakiś tam")
print(zbior1 ^ zbior2)
```

Operacje na zbiorach cz.3

- Suma dwóch zbiorów z podstawieniem
  - Zbiór można zsumować dwa zbiory korzystając z wewnętrznej metody update:

```
zbior = set("jakiś tam tekst")
print(zbior)
zbior.update("nowy tekst do zbioru")
print(zbior)
```

  - Ten sam efekt można uzyskać znacznie krótszym zapisem, korzystając z operatora |=.
  - Przy użyciu operatora |= konieczne jest utworzenie zbioru z elementów tekstu za pomocą set, czego nie trzeba było robić przy użyciu metody wewnętrznej update.

```
zbior = set("jakiś tam tekst")
print(zbior)
zbior |= set("nowy tekst do zbioru")
print(zbior)
```

- Różnica symetryczna dwóch zbiorów z podstawieniem
  - Pierwszy sposób, wykorzystujący wewnętrzną metodę difference_update:

```
zbior = set("jakiś tam tekst")
print(zbior)
zbior.symmetric_difference_update("nowy tekst do zbioru"). print(zbior)
```

  - Ten sam wynik można uzyskać przy wykorzystaniu operatora ^=. konieczne jest rzutowanie na set.

```
zbior = set("jakiś tam tekst")
print(zbior)
zbior ^= set("nowy tekst do zbioru")
print(zbior)
```

Kolekcja niezmienna

- Kolekcja niezmienny (Frozen Set) to zbiór unikalnych elementów, który jest niezmienny.

```
kolekcja = frozenset ({1,1,2,2,3,3,4,4,5,5})
print (kolekcja)                #{1, 2, 3, 4, 5}
```

- Jeśli kolekcja nie wymaga modyfikacji, korzystne jest użycie funkcji `frozenset()`, aby utworzyć kolekcję niezmienną. Niezmienne zbiory są bardziej wydajne i mogą być używane jako klucze w słownikach.
- Edycja kolekcji niezmiennej
  - Nie można do nich dodawać ani usuwać elementów.
  - Próba edycji kolekcji niezmiennej pokaże błąd

```
kolekcja.add (7)
    ^^^^^^^^^^^^

AttributeError: 'frozenset' object has no
attribute 'add'
```

Ćwiczenia część 1

1. Utwórz 2 kolekcje (sety). Jedna ma zawierać cyfry parzyste, a druga nieparzyste.
   a) Wczytuj liczbę, pobierz jej ostatnią cyfrę i przypisz do którejś z kolekcji : nieparzysty lub parzysty.
   b) Na tej podstawie oceń parzystość liczby.
2. Utwórz zbiór z liczb: 1, 2, 2, 3, 4, 4, 5.
   a) Wypisz zbiór i wyjaśnij, dlaczego ma mniej elementów niż lista.
   b) Sprawdź, czy liczba 6 należy do kolekcji
   c) Policz liczbę elementów w kolekcji
3. Dane są dwa zbiory: **A = {1, 2, 3, 4} B = {3, 4, 5,** 6}. Oblicz:
   a) sumę zbiorów
   b) część wspólną
   c) różnicę A – B
4. Z listy: **lista = [1, 2, 2, 3, 4, 4, 5]** usuń duplikaty używając **set**, a następnie zamień wynik z powrotem na listę.

Ćwiczenia część 2

5. Sprawdź, czy zbiór: **A = {1, 2}** jest podzbiorem kolekcji: **B = {1, 2, 3, 4}**.
6. Wczytaj dwa zdania od użytkownika. Utwórz kolekcje znaków i wypisz wspólne litery (bez powtórzeń).
7. Dane są dwie kolekcje uczniów zapisanych na zajęcia:
   a) **matematyka = {"Ania", "Kasia", "Piotr"}**
   b) **informatyka = {"Piotr", "Tomek", "Kasia"}**
   c) Wypisz:
      a) uczniów zapisanych na oba przedmioty
      b) uczniów zapisanych tylko na jeden przedmiot
8. Napisz program, który sprawdzi, czy dwa słowa są **anagramami**, używając kolekcji
9. Napisz program, który sprawdzi, czy dane słowo jest **heterogramem**, używając kolekcji
10. Sprawdź, czy dwa zbiory są równe, niezależnie od kolejności elementów.

Powtórzenie

1. Co to jest kolekcja?
2. Czym się różni od listy?
3. Jak utworzyć pustą kolekcję?
4. Jak zrobić kolekcję z listy?
5. Czym się wyróżnia sposób wyświetlania kolekcji poleceniem **print**?
6. Jak dodajemy nowe elementy do kolekcji?
7. Co się stanie gdy nastąpi próba dodania do kolekcji elementu już w niej występującego?
8. Jak usuwać element z kolekcji?
9. Jakie polecenie usuwa i pobiera jednocześnie element z kolekcji?
10. Jak sprawdzić, czy dany element należy do kolekcji?
11. Jak wyświetlić zawartość kolekcji?
12. Jak zmodyfikować kolekcję?
13. Jak zrealizować operacje na zbiorach:
    a) łączenie,
    b) część wspólna,
    c) różnica,
    d) różnica symetryczna,
    e) suma z podstawieniem,
    f) różnica symetryczna z podstawieniem.
14. Jak się tworzy kolekcję niezmienną?