Języki programowania – Python
Standardowe kontenery i ich obsługa

Marzena Nowakowska

Wydział Zarządzania i Modelowania Komputerowego
Politechnika Świętokrzyska

Budynek C, p. 3.21

spimn@tu.kielce.pl

Mutowalność

Dana **niemutowalna** to taka, której wartości nie można zmienić „w miejscu” jej położenia (na stosie). Należą do nich: **liczby**, **wartości logiczne**, **łańcuchy**, **zakresy (*range*)** i **krotki**.

Dane prawie wszystkich innych typów są mutowalne. Dana **mutowalna** to taka, której wartość można zmienić „w miejscu” jej położenia (na stosie). Są to np.: **listy**, **słowniki**, **zbiory**, **obiekty** zdefiniowane przez projektanta (programistę).

Kontenery

Kontener to obiekt takiego typu, który składa się z elementów; przechowuje dowolną liczbę innych obiektów. Kontenery zapewniają sposób **dostępu do zawartych w nich obiektów** i **iterowanie po nich**. Rozmaite typy kontenerów różnią się sposobem dostępu do elementów, modyfikowalnością oraz optymalizacją pod względem różnych zastosowań.

Kontenery

Sekwencje (dostęp do elementu poprzez indeks):
- Łańcuchy
- Krotki
- Listy

Nie sekwencje:
- Słowniki (dostęp do wartości poprzez klucz)
- Zbiory (brak dostępu wprost do elementu, tylko poprzez operacje na zbiorze)

Wybrane metody dla kontenerów standardowych

- Lista (**list**)

  append(), copy(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()
- Krotka (**tuple**)

  count(), index()
- Łańcuch (**str**)

  capitalize(), count(), find(), index(), join(), lower(), replace(), split(), stratswith(), upper()
- Słownik (**dict**)

  clear(), copy(), items(), keys(), pop(), values()
- Zbiór (**set**)

  add(), copy(), issuperset(), remove()

Wybrane przydatne funkcje

len() range() slice(), sum() min() max() zip()

Funkcje pracują na kontenerach.

Typ zwracanego wyniku zależy od funkcji i jej argumentów.

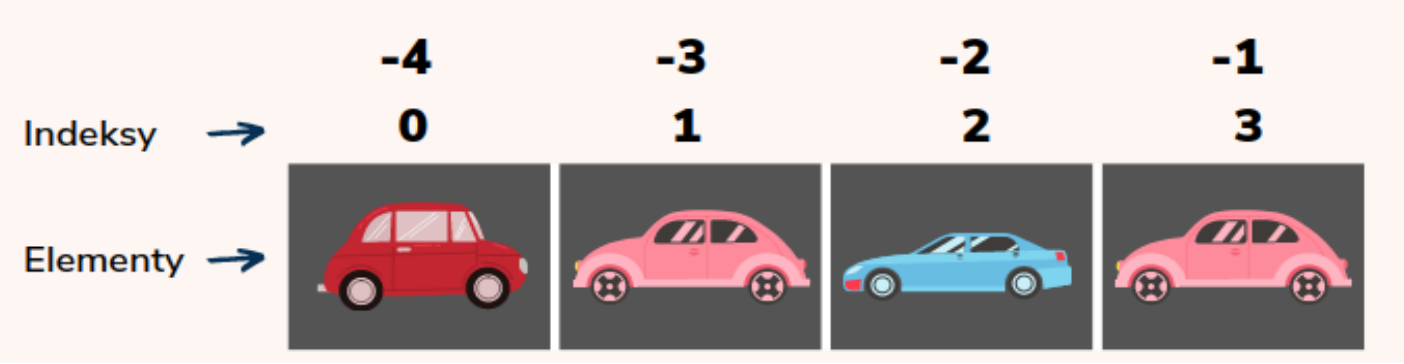
Sekwencyjność

Sekwencje Pythona to uporządkowane pozycyjnie kolekcje elementów. Do dowolnego elementu w sekwencji można się odwołać używając jego numeru (indeksu), przy czym numeracja zaczyna się od zera. Pierwszy element ma więc numer 0, drugi – numer 1, itd. Odwołanie do elementu sekwencji jest następujące:

nazwa_sekwencji[*numer_elementu*]

Jeżeli sekwencja *s* ma *n* elementów, to przykładowe odwołania do nich są następujące: s[0], s[1], s[n-1] ← to jest ostatni element sekwencji.

Sekwencje Pythona: **łańcuchy**, k**rotki**, **listy**



Źródło: Internet

Wybrane operacje na sekwencjach (przykłady dla listy)

- Tworzenie obiektu pustego

  Przykład: a1 = [] ; a11 = list()

- Testowanie członkostwa, czyli *in* oraz *not in* w wyrażeniach. Wynik jest typu *bool*.

  Przykład: 0 in [0,1,2,3,4] → True

- Indeksowanie, czyli odwołanie do elementu sekwencji poprzez wskazanie numeru (indeksu). Składnia: *obiekt*[*indeks*]

  Przykład: [4, 3, 2, 1][0] → 4

- Wycinanie (pobieranie podzbioru elementów na podstawie ich indeksów). Uzyskanie fragmentu sekwencji; typ wyniku jest dziedziczony. Składnia: [*początek* : *koniec* : *krok*]

  Wartości domyślne: *początek* = 0, *koniec* = nr ostatniego+1, *krok* = 1

  Przykład: [0,1,2,3,4][1::2] → [1, 3]

- Porównanie; zaczyna się od pierwszego elementu z każdej sekwencji, jeżeli są równe, to porównywane są następne dwa elementy itd., aż znajdą się elementy, które różnią się między sobą; wtedy ta lista jest większa, która ma większy z tych dwóch elementów różniących się (dalej nie ma już sprawdzania). Jeżeli elementy obu list są sobie równe, listy są równe. Wynik jest typu *bool*.

  Przykład: [0,1,2,3,4] < [-1, 2, 3] → False

- Multiplikacji obiektu. Typ wyniku jest dziedziczony. Składnia: *obiekt * liczba_naturalna*

  Przykład: [0,1,2,3,4] *2 → [0,1,2,3,4,0,1,2,3,4]

- Dodawanie (składanie, sklejanie) obiektów. Typ wyniku jest dziedziczony. Składnia: *obiekt1 + obiekt2*

  Przykład: ['a', 1,2,'b'] + [100, 45] → ['a', 1, 2, 'b', 100, 45]

Iterator, iteracja, iterowalność

Iterator – obiekt, który zawiera policzalną liczbę wartości. Jest wykorzystywany do sekwencyjnego dostępu do wszystkich elementów zawartych w innym obiekcie, który jest iterowalny, zwykle kontenerze. Iterator jest czasem nazywany kursorem. Podstawowym celem iteratora jest pozwolić użytkownikowi przetworzyć każdy element w kontenerze bez konieczności zagłębiania się w jego wewnętrzną strukturę.

Iterator obsługuje proces **iterowania** (**iteracji**), czyli przechodzenia po elementach jakiegoś zbioru obiektów. Jest czynnością powtarzania tej samej operacji w pętli określoną liczbę razy lub do spełnienia określonego warunku. Mianem iteracji określa się też operacje wykonywane wewnątrz takiej pętli.

Obiekt iterowalny (iterable, iterowalny – jako rzeczownik) to taki, po którym można iterować. Generuje on iterator po przekazaniu obiektu do metody *iter*(). Iterator iteruje po obiekcie iterowalnym przy użyciu swojej metody __*next*__(), która zwraca następny element obiektu. Każdy iterator jest obiektem iterowanym, ale nie każdy obiekt iterowalny jest iteratorem, np. lista nie jest iteratorem.

Standardowe obiekty iterowane w Pythonie: lista, krotka, łańcuch, słownik (tu iterowanie jest realizowane domyślnie po kluczach), zbiór.
Funkcje zwracające iteratory: range(), zip(), map(), filter(), enumarate(), reversed().

Instrukcja pętli *for*

for ***el*** **in** ***obiekt_iterowalny*****:**
    ciąg_instrukcji1_w bloku_for
[else:
    ciąg_instrukcji2_w_bloku_else**]**

```
# Pętla for dla listy;
# Obliczanie średniej geometrycznej

from math import pow
dane=[2,5,3,11,6]
stp=len(dane)
iloczyn=1
for el in dane:
    iloczyn*= el
sg1=pow(iloczyn, 1/stp)
sg2=iloczyn**(1/stp)
```

Instrukcje bloku dla słowa ***for*** (ciąg instrukcji po słowie *for*) są wykonywane dla każdego *el*, który przyjmuje wartości zdefiniowane przez *obiekt_iterowalny*.
Instrukcje bloku dla słowa ***else*** są wykonywane, jeśli elementy *obiektu_iterowalnego* użyte w pętli *for* się wyczerpią i blok *for* nie zostanie ani razu wykonany.
Ważne, by wszystkie instrukcje obu bloków zapisywane były z takim samym wcięciem.

Przyklady dla pętli *for*

```
# Sprawdzenie, czy w liście jest co najmniej
# jedna liczba parzysta; instrukcja break
# oraz element else

liczby = [-1, 1, 5, 7, 9]
parz=0
for i in liczby:
    if i%2 ==0:
        print("Na liście są liczby parzyste");
        break;
else:
    print("Na liście nie ma liczb parzystych")
```

```
# Pętla for dla słownika;
# Wyświetlenie kolejnych elementów

d={'a':1, 'b':2,'c':3}
for klucz in d:
    print(klucz, "=>", d[klucz])
print()  # pusty wiersz
for (klucz, wartosc) in d.items():
    print(klucz, "==>", wartosc)
```

```
# Tabliczka mnożenia; zagnieżdżenie pętli for

for i in range(0, 10, 1) :
    for j in range (0, 10, 1):
        print("%3i x %3i = %4i %s" % (i, j, i*j, "\n" if j==9 else ""))
```

Iterator z funkcji *range*

Do tworzenia obiektu iterowalnego bardzo często stosuje się funkcję *range*, za pomocą której definiuje się liczbę iteracji do wykonania:

range([*start*,] *stop* [, *krok*])

gdzie parametry ***start*** i ***krok*** są opcjonalne; pominięcie *start* jest równoznaczne z przyjęciem wartości startowej 0, pominięcie parametru *krok* oznacza przyjęcie kroku (odległości między elementami sekwencji) równego 1; parametr ***stop*** jest ograniczeniem sekwencji, już **do niej nie należy**.

Przykłady

range(10)	generuje sekwencję:	0,1,2,3,4,5,6,7,8,9
range (1,10)	generuje sekwencję:	1,2,3,4,5,6,7,8,9
range (1,10,2)	generuje sekwencję:	1,3,5,7,9
range (9,0,-2)	generuje sekwencję:	9,7,5,3,1

Przykład dla *for*

```
from math import sin, pi
print("Tablica funkcji sinus w przedziale 90-100 stopni z krokiem 1 stopień")
print("%3s\t%8s" %("x", "sin(x)"))
for i in range (90,101, 1):
    print("%3i\t%8.4f" % (i, sin(i*pi/180)))
```

Instrukcja pętli *while*

while ***wyrażenie_logiczne*****:**
    ciąg_instrukcji1_w bloku_while
[else:
    ciąg_instrukcji2_w_bloku_else**]**

Instrukcje bloku *while* powtarzane są tak długo, jak długo prawdziwy jest *warunek*.
Część *else* jest wykonywana, jeśli warunek w pętli *while* ma wartość *False*; po czym pętla *while* się kończy.
Tak jak w przypadku pętli *for* wszystkie instrukcje obu bloków: *while* i *else* są zapisywane z takim samym wcięciem.

Instrukcja *break* w pętlach *for* i *while*

for ***el*** **in** ***obiekt_iterowalny*****:**
    Ciąg_instrukcji _w_ obrębie_pętli
    if ***warunek:***
        break
    Ciąg_instrukcji _w_ obrębie_pętli
Ciąg_dalszy_programu; czyli
Ciąg_instrukcji_po_zakończeniu_pętli

while ***wyrażenie_logiczne*****:**
    Ciąg_instrukcji _w_ obrębie_pętli
    if ***warunek:***
        break
    Ciąg_instrukcji _w_ obrębie_pętli
Ciąg_dalszy_programu; czyli
Ciąg_instrukcji_po_zakończeniu_pętli

```
max_l_prob=1205
licznik=0
liczba_1=int(input("Podaj liczbę calkowitą: "))
from random import randrange
liczba_2=randrange(liczba_1+1)
while liczba_1 !=liczba_2:
    if licznik > max_l_prob:
        break;
    licznik+=1
    liczba_2=randrange(liczba_1+1)
print("Liczba prób:", licznik)
print("Liczba zadana:", liczba_1)
print("Ostatnia liczba wygenerowana:", liczba_2)
```

Instrukcja *continue* w pętlach *for* i *while*

for ***el*** **in** ***obiekt_iterowalny*****:**
Ciąg_instrukcji _w_ obrębie_pętli
if ***warunek:***
continue
Ciąg_instrukcji _w_ obrębie_pętli
Ciąg_dalszy_programu; czyli
Ciąg_instrukcji_po_zakończeniu_pętli

while ***wyrażenie_logiczne*****:**
Ciąg_instrukcji _w_ obrębie_pętli
if ***warunek:***
continue
Ciąg_instrukcji _w_ obrębie_pętli
Ciąg_dalszy_programu; czyli
Ciąg_instrukcji_po_zakończeniu_pętli

```
# Zadanie z instrukcją continue w pętli while
x = [5, 1, 10, 1, 15, 1, 20, 1, 25, 1, 30, 1, 35]
n=len(x)
l=-1
suma=0
while l < n-1:
    l+=1
    if x[l] % 5 == 0:
        continue
    suma+=x[l]
 print("Suma liczb jakich?:", suma)
```

Złożenie (*comprehension*) w Pythonie

Python umożliwia tworzenie nowego obiektu iterowalnego przy wykorzystaniu podanej sekwencji (jednej lub kilku). Taka operacja nazywa się **złożeniem** (*comprehension*).
Jest to sposób napisania zwięzłego bloku kodu w celu wygenerowania obiektu, który może być listą, słownikiem, zbiorem lub generatorem. Może obejmować wiele etapów konwersji różnych typów sekwencji. Składnia złożenia:

Nawias *el_definiowany* **for** ***el_definiujący*** **in** *ob_iterowalny* ***Nawias***

gdzie *Nawias* jest znakiem właściwym dla danego kontenera.

- Złożenie dla list: *ls1 = [i for i in range(5)] ; ls2 = [j**2 for j in ls1]*
- Złożenie dla krotek: *ls11 = tuple(i for i in range(5)) ; ls22 = tuple(j**2 for j in ls1)*
- Złożenie dla zbiorów: *ls111 = {i for i in range(5)} ; ls222 = {j**2 for j in ls111}*
- Złożenie dla słownika:

```
s1 = [nr for nr in range(1, 8,1)] ;
s2 = ['Pn','Wt','Śr','Czw','Pt', 'Sb', "Nd"]
dni_tyg ={nr:nazwa for (nr, nazwa) in zip(s1, s2)}
```

```
# Złożenie w rozwiązaniu zadania z instrukcją continue w pętli while
x = [5, 1, 10, 1, 15, 1, 20, 1, 25, 1, 30, 1, 35]
suma=sum([el if el%5 != 0 else 0 for el in x])
print("Suma liczb jakich?:", suma)
```