Rafał Korzeniewski

Wstęp do języka Python. Część 7

Słowniki i zbiory to kolejne narzędzia w naszym programistycznym niezbędniku. W artykule poznamy ich szczególne cechy, nauczymy się je tworzyć, modyfikować, wykorzystywać. Struktury te, szczególnie słowniki, mają bardzo wszechstronne zastosowania, dlatego warto im się bliżej przyjrzeć.

12+

DOWIESZ SIĘ

- Jak korzystać ze słowników oraz zbiorów w Pythonie.

POTRZEBNA WIEDZA

- Jak tworzy się zmienne w Pythonie.

SŁOWNIK

Do tej pory poznaliśmy struktury takie jak listy i krotki, które umożliwiają nam przechowywanie różnych obiektów i dostęp do nich poprzez indeks – czyli pozycję danego elementu w kolekcji. Jak pamiętamy, indeksy takie to kolejne liczby naturalne (0, 1, 2, 3, ...), a liczenie rozpoczyna się od 0. W niektórych sytuacjach wygodniej by było jednak nie posługiwać się takim indeksem liczbowym, ale na przykład jakąś nazwą. Powiedzmy, że chcemy mieć strukturę opisującą jakąś osobę. Na początek użyjmy do tego listy:

Listing 1. Lista przedstawiająca osobę — jej imię i nazwisko

```
In [1]: osoba = ["Andrew", "Wiggin"]

In [2]: print(osoba[0])
Andrew
In [3]: print(osoba[1])
Wiggin
```

Jak widzimy w Listingu 1, indeks imienia to `0`, a indeks nazwiska to `1`. Dosyć łatwo tutaj rozpoznać, co jest imieniem, a co nazwiskiem.

Załóżmy teraz, że chcielibyśmy opisać inną osobę. Przychodzi mi tu na myśl mój przyjaciel – uzdolniony trębacz i właściciel sklepu z instrumentami dętymi w Bydgoszczy. Nazywa się on Zbigniew Zygmunt. Prawdę mówiąc, przez długi czas mieszało mi się w głowie, co jest imieniem, a co nazwiskiem. Gdyby ktoś zobaczył taką listę jak poniżej, to też mógłby mieć z tym zadaniem problem:

```
In [4]: osoba = ["Zbigniew", "Zygmunt"]
```

Oczywiście moglibyśmy zaplanować, że zawsze imię jest pierwsze i się potem tego trzymać. Ale nietrudno tutaj o pomyłki. A gdyby tak opisać tę osobę trochę inaczej? Na zasadzie klucz i wartość. Coś jak w tabeli poniżej:

KLUCZ (ANG. KEY)	WARTOŚĆ (ANG. VALUE)
imię	Zbigniew
nazwisko	Zygmunt

Tabela. 1. Przedstawienie osoby w formie tabeli

W Pythonie do zapisu takich danych możemy użyć słownika (ang. *dict*). Jest kilka sposobów, by go utworzyć. Najczęściej wykorzystujemy do tego nawiasy klamrowe {} – tak jak przedstawiono w Listingu 2. Natomiast do wybierania elementów służą nawiasy kwadratowe [ ], podobnie jak przy wybieraniu elementów z wcześniej

poznanych struktur. Tym razem jednak zamiast liczby reprezentującej indeks wstawiamy klucz.

Listing 2. Wykorzystanie słownika do opisu osoby

```
In [6]: osoba = {
    "imię": "Zbigniew",
    "nazwisko": "Zygmunt"
}

In [7]: osoba["imię"]
Out[7]: 'Zbigniew'

In [8]: osoba["nazwisko"]
Out[8]: 'Zygmunt'
```

Innym sposobem będzie wykorzystanie konstruktora `dict()` i podanie w nim argumentów z przypisaniem do nich wartości tak jak w Listingu 3.

Listing 3. Tworzenie słownika przy pomocy „dict"

```
In [9]: ender = dict(imię="Andrew",
nazwisko="Widding")

In [10]: ender
Out[10]: {'imię': 'Andrew',
'nazwisko': 'Widding'}
In [11]: type(ender)
Out[11]: dict
```

Jak widzimy, `dict` to też nazwa tego typu. A więc mamy tu podobieństwo do wcześniej poznanych `list`, `tuple`, które w poprzednim numerze PJR nazywaliśmy dla uproszczenia funkcjami.

Przydatnym wykorzystaniem tego konstruktora jest tworzenie słownika na podstawie listy lub tupli zawierającej w sobie pary wartości. Te pary mogą być znowu listą albo krotką (inna nazwa to tupla) – tak jak w Listingu 4.

Listing 4. Tworzenie słownika z listy par wartości

```
In [12]: osoba = [["imię", "George"],
["nazwisko","Dyson"]]

In [13]: dict(osoba)
Out[13]: {'imię': 'George',
'nazwisko': 'Dyson'}

In [14]: dict([("imię", "George"),
("nazwisko","Dyson")])
Out[14]: {'imię': 'George', 'nazwisko': 'Dyson'}
```

Co się stanie, jeśli zamiast pary podamy 3 wartości? Wystarczy, że dodasz do którejś z wewnętrznych list kolejny element. Python powinien cię poinformować, że coś poszło nie tak, na przykład takim komunikatem:

```
ValueError: dictionary update sequence element
#0 has length 3; 2 is required
```

Warto robić takie eksperymenty. Poznawać komunikaty błędów. Taka wiedza pozwoli ci w przyszłości lepiej rozumieć i naprawiać napotkane problemy.

W listach i tuplach każdy indeks jest unikalny, co oznacza, że może wystąpić tylko raz. Wartości mogą być natomiast dowolne i mogą się powtarzać. Jak jest w słowniku? Czy możemy mieć dwa takie same klucze?

Listing 5. Powtarzający się klucz

```
In [18]: {"a": 10, "b":20, "a":30}
Out[18]: {'a': 30, 'b': 20}
```

Błędu co prawda nie ma, ale jak widać, klucz w wyniku występuje tylko raz. Tworząc słowniki, powinniśmy więc pamiętać o tym, by nie nadpisać potrzebnej nam wartości.

Pamiętasz, jak tworzyło się pustą listę? Można to zrobić albo przy pomocy nawiasów `[]`, albo wywołania konstruktora `list()`. Analogicznie będzie w przypadku słownika:

Listing 6. Tworzenie pustego słownika

```
In [19]: {}
Out[19]: {}

In [20]: dict()
Out[20]: {}
```

Do istniejącego słownika możemy dodawać nowe klucze i wartości. Tak jak w Listingu 7:

Listing 7. Dodawanie nowych kluczy i wartości do słownika

```
In [1]: osoba = [("imię", "George") ,
("nazwisko","Dyson")]
```

```
In [2]: osoba = dict(osoba)

In [3]: osoba
Out[3]: {'imię': 'George', 'nazwisko': 'Dyson'}

In [4]: osoba["zawód"] = "pisarz"

In [5]: osoba["rok urodzenia"] = 1953

In [6]: osoba
Out[6]:
{'imię': 'George',
 'nazwisko': 'Dyson',
 'zawód': 'pisarz',
 'rok urodzenia': 1953}
```

W ten sam sposób, w który wartości dodajemy, możemy też je nadpisywać. Jak widać, w kluczu `"rok urodzenia"` zawarta jest spacja. Takiej spacji nie możemy wstawić wtedy, gdy tworzymy słownik, wywołując `dict` z podaniem argumentów:

Listing 8. Błąd składni przy tworzeniu słownika

```
In [7]: dict(rok urodzenia=1953)
  File "<ipython-input-7-312a61ae44fb>", line 1
    dict(rok urodzenia=1953)
               ^
SyntaxError: invalid syntax
```

Wiemy już, że klucze muszą być unikalne. Czy są jeszcze jakieś ograniczenia na nie nałożone? Czy klucz może być tylko napisem, czy czymś jeszcze?

Listing 9. Przykłady różnych kluczy

```
In [8]: {1: "A"}
Out[8]: {1: 'A'}

In [9]: {2.2: "A"}
Out[9]: {2.2: 'A'}

In [10]: {("a", "b"): "C"}
Out[10]: {('a', 'b'): 'C'}

In [11]: {False: "false"}
Out[11]: {False: 'false'}

In [12]: {None: "nic"}
Out[12]: {None: 'nic'}

In [13]: {[1, 2]: "lista"}
---------------------------------------------
TypeError Traceback (most recent call last)
<ipython-input-13-d1282d4f7522> in <module>
----> 1 {[1, 2]: "lista"}

TypeError: unhashable type: 'list'

In [14]: {print: "printuje"}
Out[14]: {<function print>: 'printuje'}
```

Jak widać w Listingu 9, obiekty różnego typu mogą być kluczem. Jednak nie wszystkie. Mogą być nim liczby, napisy, wartości typu `bool`, `None`, a nawet tupla czy jakaś funkcja. Ale nie może nim być lista. Kluczem nie może być też inny słownik. Pamiętacie, jak wspominałem kiedyś, że niektóre obiekty są mutowalne, a inne nie? To znaczy, że niektóre obiekty mogą być zmieniane, a inne przez cały czas swego istnienia pozostają takie same. Napisów, liczb czy tupli nie można zmienić – są więc niemutowalne. Natomiast listy, słowniki i zbiory, które za chwilę poznamy, można zmieniać. W codziennej praktyce, z dużym przybliżeniem, możemy przyjąć, że kluczem mogą być typy niemutowalne. Tak naprawdę chodzi o trochę inną cechę tych obiektów – tak zwaną hashowalność (czytaj: haszowalność), co po angielsku zapisujemy: *hashable*. Jeśli obiekt jest hashowalny, to ma specjalną wartość – hash, która nie zmienia się przez całe życie obiektu. W przyszłości nauczymy się dodawać obiektom taką cechę.

Wartością w słowniku może być w zasadzie dowolny inny obiekt. Może to być więc napis, liczba, tupla, lista, a nawet inny słownik. W ramach ćwiczenia możesz spróbować samodzielnie utworzyć słownik superbohaterów. Kluczem w takim słowniku niech będzie nazwa, na przykład Superman, Spiderman, Hulk. Wartościami natomiast niech będą inne słowniki, w których możesz zawrzeć opis danej osoby. Kluczami mogą być wtedy prawdziwe imię, nazwisko, wiek, waga, opis, supermoce. Spróbuj też samodzielnie wybierać takie wartości ze słownika. W przyszłości na pewno będziemy tworzyć takie bardziej zaawansowane struktury.

Jakie metody ma słownik? Co możemy z nim zrobić? W ustaleniu tego pomoże nam funkcja `dir`.

Listing 10. Metody słownika

```
In [15]: dir({})
Out[15]:
[ ...
 'clear',
```

```
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

Nie będziemy omawiać wszystkich metod, a jedynie niektóre. Przypominam o użyciu funkcji `help`, która pozwoli przeczytać dokumentację funkcji, na przykład tak jak w Listingu 11.

Listing 11. Pomoc dla metody „keys" słownika

```
In [16]: help({}.keys)

Help on built-in function keys:

    (...) method of builtins.dict instance
    D.keys() -> a set-like object
providing a view on D's keys
(END)
```

W Listingu 12 pokazano przykłady działania kilku metod:

Listing 12. Działanie wybranych metod

```
In [17]: osoba.keys()
Out[17]: dict_keys(['imię',
'nazwisko', 'zawód', 'rok urodzenia'])

In [18]: osoba.values()
Out[18]: dict_values(['George', 'Dyson',
'pisarz', 1953])

In [19]: osoba.items()
Out[19]: dict_items([('imię', 'George'),
('nazwisko', 'Dyson'),
('zawód', 'pisarz'), ('rok urodzenia', 1953)])

In [20]: osoba.pop('rok urodzenia')
Out[20]: 1953

In [21]: osoba
Out[21]: {'imię': 'George', 'nazwisko': 'Dyson',
'zawód': 'pisarz'}

In [22]: osoba.popitem()
Out[22]: ('zawód', 'pisarz')
```

Jak widzimy

- `keys` – zwraca obiekt zawierający same klucze,
- `values` – zwraca same wartości; to tak, jakby z Tabeli 1 wziąć tylko kolumnę wartości,
- `items` – zwraca listę par: klucz, wartość,
- `pop` – zwraca wartość dla zadanego klucza i usuwa go ze słownika,
- `popitem` – zwraca ostatnio dodaną parę.

W przypadku list, tupli, napisów w sytuacji, gdy próbowaliśmy pobrać jakiś element spoza zakresu, dostawaliśmy `IndexError`. Co się stanie, gdy spróbujemy wybrać ze słownika wartość dla klucza, którego w nim nie ma?

Listing 13. Próba pobrania wartości dla nieistniejącego klucza

```
In [1]: pol_ang = {"pies": "dog", "kot": "cat"}

In [2]: pol_ang["krowa"]
---------------------------------------
KeyError Traceback (most recent call last)
<ipython-input-2-fe0417062d52> in <module>
----> 1 pol_ang["krowa"]

KeyError: 'krowa'
```

Jak widać, tu też dostajemy błąd. Tym razem jest to `KeyError`. Są różne sposoby radzenia sobie z taką sytuacją. Możemy na przykład najpierw sprawdzić, czy klucz znajduje się w słowniku, przy pomocy wyrażenia `klucz in slownik`. Kiedy zależy nam na tym, by zawsze była zwracana jakaś wartość, warto użyć metody `get`.

WARTO WIEDZIEĆ

Słowniki od Pythona w wersji 3.6 sa kolekcjami uporządkowanymi. To znaczy, że zachowywana jest kolejność dodawania elementów. W starszych wersjach Pythona były to kolekcje, które takiego porządku nie miały. Elementy były rozkładane według specjalnego algorytmu, który zapewniał szybki dostęp do każdego elementu – bez względu na to, czy znajdował się on na końcu, czy początku kolekcji. Z czasem udało się zastosować taki algorytm, który zapewniał zarówno szybki dostęp, jak i utrzymanie porządku dodawania.

Listing 14. Sprawdzenie, czy klucz zawiera się w słowniku, oraz użycie metody „get"

```
In [2]: "krowa" in pol_ang
Out[2]: False

In [3]: if "krowa" in pol_ang:
   ...:     print(pol_ang["krowa"])
   ...:

In [4]: wartosc = pol_ang.get("krowa")

In [5]: print(wartosc)
None

In [6]: wartosc = pol_ang.get("krowa",
"Brak tłumaczenia")

In [7]: wartosc
Out[7]: 'Brak tłumaczenia'
```

Jak widać w Listingu 14, do metody get przekazujemy klucz. Jeśli taki klucz istnieje w słowniku, to zostanie zwrócona dla niego wartość. W przeciwnym razie zwracane jest None, lub wartość domyślna, która jest opcjonalnym argumentem tej metody.

Możesz teraz spróbować użyć tej metody w następującym zadaniu:

Przy pomocy pętli policz, ile razy każdy znak występuje w napisie „konstantynopolitańczyneczka". Do przechowywania informacji o częstotliwości użyj słownika.

CIEKAWOSTKA

Nazwy użyte w listingach nawiązywały do różnych fikcyjnych i rzeczywistych postaci. „Ender" to postać ze znakomitych książek Orsona Scotta Carda. Po raz pierwszy pojawiła się w opowiadaniu „Gra Endera" opublikowanym w 1977 roku. W 1985 r. autor rozbudował te opowiadanie do formy powieści, a w kolejnych latach wydawał dalsze książki i opowiadania, które składają się na rozbudowaną „Sagę Endera". Co ciekawe, tej fikcyjnej postaci autor nadał polskie korzenie. Jego ojciec był emigrantem z Polski, który zanim przyjął nazwisko John Paul Wiggin nazywał się Jan Paweł Wieczorek. George Dyson to z kolei postać z innej książki. Tym razem nie jest to postać fikcyjna. George to syn wybitnego fizyka – Freemana Dysona. Fragment ich barwnych życiorysów opisany jest w książce „Kosmolot i Czółno" Kennetha Browera.

Trzeba więc utworzyć pusty słownik na takie zliczenia i przypisać go do jakiejś zmiennej. Do innej zmiennej można przypisać napis. Następnie trzeba pętlą for przejść po znakach w tym napisie i powiększać w słowniku wartość licznika. Starą wartość można powiązać z nową w sposób podobny do tego z Listingu 15:

Listing 15. Przykład ustawienia nowe wartości dla klucza przy wykorzystaniu starej

```
In [10]: drukarka = {"kartki": 500 }

In [11]: drukarka["kartki"] = drukarka["kartki"] - 1

In [12]: drukarka
Out[12]: {'kartki': 499}
```

Jak widzimy, w jednej linii odwołujemy się do klucza dwukrotnie. Działa to w ten sposób, że wyliczane jest najpierw wyrażenie po prawej stronie znaku równości i ta nowa wartość wstawiana jest pod ten sam klucz w słowniku, co można w krokach przedstawić następująco:

```
1. drukarka["kartki"] = drukarka["kartki"] - 1
2. drukarka["kartki"] = 500 - 1
3. drukarka["kartki"] = 499
```

Skoro słownik to kolekcja, to czy można iterować po nim pętlą for tak jak w przypadku wcześniejszych kolekcji? Jak najbardziej. Domyślnie taka iteracja przebiega po kluczach:

Listing 16. Iterowanie po kluczach

```
In [13]: for slowo in pol_ang:
    ...:     print(slowo)
    ...:
pies
kot
```

Możemy jednak łatwo iterować też po parach:

Listing 17. Iterowanie po parach (items)

```
In [14]: for item in pol_ang.items():
    ...:     print(item)
    ...:
('pies', 'dog')
```

```
('kot', 'cat')

In [15]: for pol, ang in pol_ang.items():
    ...:     print(pol, ang, sep=" - ")
    ...:
pies - dog
kot - cat
```

To tyle podstawowej wiedzy o słownikach. W przyszłości zapewne spotkacie się z różnymi ich wariantami takimi jak na przykład `defaultdict` z modułu `collections`.

Zbiory

Zbiory to kolejna ciekawa struktura, która ma unikalne cechy. Tworzymy je podobnie jak słowniki przy pomocy nawiasów klamrowych, przy czym tutaj nie ma par, a pojedyncze elementy. Możemy też utworzyć zbiór, wywołując konstruktor `set()`. Możemy mu przekazać jakiś iterowalny obiekt – na przykład napis, listę, tuplę, a nawet słownik. Oba sposoby pokazano w Listingu 18.

Listing 18. Tworzenie zbioru

```
In [1]: A = {1, 2, 3, 4}

In [2]: B = set([3, 4, 5, 6])
```

Do tej pory puste listy i słowniki mogliśmy tworzyć przy pomocy nawiasów – odpowiednio [ ] i { }. By utworzyć pusty zbiór, musimy wywołać funkcję `set()` bez podawania argumentów.

Zbiór – podobnie jak klucze w słowniku – zawierać może tylko unikalne elementy:

Listing 19. Zbiór zawiera tylko unikalne elementy

```
In [2]: A = {1, 2, 3, 4, 1, 2, 3, 4}

In [3]: A
Out[3]: {1, 2, 3, 4}
```

Zamiana kolekcji na zbiór jest więc wygodnym sposobem na to, by otrzymać zbiór unikalnych elementów w kolekcji.

Wartości w zbiorze są nieuporządkowane – w tym sensie, że nie zależą od kolejności dodawania. Za ich rozmieszczenie odpowiada specjalny algorytm:

Listing 20. Kolejność dodawania nie jest kolejnością przechowywania

```
In [4]: C = {"A", 1, "B", 2, "C", 0}

In [5]: C
Out[5]: {0, 1, 2, 'A', 'B', 'C'}
```

Przy operacjach na zbiorach wprowadzane są nowe operatory. Pozwalają one na wykonanie znanych z matematyki operacji na zbiorach:

Jeśli mamy dwa zbiory:

$$A = \{1, 2, 3, 4\}$$
$$B = \{3, 4, 5, 6\}$$

to:

- sumą zbiorów A i B nazywamy taki zbiór, którego elementy są w A lub B.

Matematycznie zapisujemy to jako:

$$A \cup B = \{1, 2, 3, 4, 5, 6\}$$

A oto przykład w Pythonie:

```
In [10]: A | B
Out[10]: {1, 2, 3, 4, 5, 6}
```

Można to też uzyskać przy pomocy metody `union`:

```
In [15]: A.union(B)
Out[15]: {1, 2, 3, 4, 5, 6}
```

- iloczynem A i B nazywamy taki zbiór, którego elementy są zarówno w A, jak i w B.

Matematycznie zapisujemy to jako:

$$A \cup B = \{3, 4\}$$

A oto przykład w Pythonie:

```
In [11]: A & B
Out[11]: {3, 4}
```

Można to też uzyskać przy pomocy metody `intersection` (przecięcie):

```
In [16]: A.intersection(B)
Out[16]: {3, 4}
```

» różnicą zbiorów A i B nazywamy taki zbiór, którego elementy są w A, ale nie ma ich w B.

Matematycznie zapisujemy to jako:

$$A \setminus B = \{1, 2\}$$

lub

$$A \quad B = \{1, 2\}$$

A oto przykład w Pythonie:

```
In [12]: A - B
Out[12]: {1, 2}
```

Można to też uzyskać przy pomocy metody `difference` (różnica):

```
In [17]: A.difference(B)
Out[17]: {1, 2}
```

» różnicą symetryczną A i B nazywamy taki zbiór, którego elementy należą do A, ale nie należą do B, oraz te elementy z B, które nie należą do A.

Matematycznie zapisujemy to jako:

$$A \oplus B = \{1, 2, 5, 6\}$$

A oto przykład w Pythonie:

```
In [13]: A ^ B
Out[13]: {1, 2, 5, 6}
```

Można to też uzyskać przy pomocy metody `symmetric_difference`:

```
In [18]: A.symmetric_difference(B)
Out[18]: {1, 2, 5, 6}
```

Aby do istniejącego zbioru dodać nowy element, należy użyć metody `add`.

```
In [19]: A.add(10)

In [20]: A
Out[20]: {1, 2, 3, 4, 10}
```

Po elementach zbioru możemy przechodzić pętlą `for` oraz sprawdzać, czy element się w nim znajduje, przy pomocy operatora `in`. Dokładnie tak samo jak w przypadku innych kolekcji.

Jak usunąć elementy ze zbioru? Postaraj się to ustalić samodzielnie, korzystając z dokumentacji.

Na ten moment to wszystko. Do zobaczenia w kolejnym numerze!

Rafał Korzeniewski

Z wykształcenia muzyk-puzonista i fizyk. Z zamiłowania i zawodu Pythonista. Trener Pythona, współorganizator PyWaw (http://pywaw.org) – warszawskiego meetupu poświęconego Pythonowi. W wolnych chwilach uczy się gry na nowych instrumentach, udziela się społecznie i dużo czyta.

KORZENIEWSKI@GMAIL.COM

ZAPAMIĘTAJ

- Słowniki zawierają pary: kluoz-wartość. Kluoze muszą być unikalne i hashowalne. Wartością może być dowolny obiekt.
- Zbiory nie mają kolejności i zawierają unikalne wartości.

ĆWICZ W DOMU

- Postaraj się samodzielnie ustalić, jak połączyć ze sobą dwa słowniki.
- Ustal, w jaki sposób sprawdzić, czy jakiś zbiór jest podzbiorem innego zbioru.