Rozdział 10

Krotki

10.1. Krotki są niezmienne

Krotka[1] jest sekwencją wartości, podobnie jak lista. Wartości zapisane w krotce mogą być dowolnego typu i są indeksowane liczbami całkowitymi. Ważną różnicą jest to, że krotki są *niezmienne*. Krotki są również *porównywalne* i *haszowalne*, więc możemy sortować ich listy i używać krotek jako kluczy w słownikach.

Składniowo krotka jest oddzieloną przecinkami listą wartości:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Chociaż nie jest to konieczne, krotki często ogranicza się nawiasami, tak aby pomóc nam szybko je zidentyfikować podczas przeglądania kodu Pythona:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Aby utworzyć krotkę z pojedynczym elementem (tzw. *singleton*), należy na końcu dołączyć przecinek:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

Bez przecinka Python traktuje `('a')` jako wyrażenie z ciągiem znaków w nawiasach, które ewaluuje się do ciągu znaków:

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
```

Innym sposobem na skonstruowanie krotki jest wbudowana funkcja `tuple()`. Bez żadnego argumentu, tworzy ona pustą krotkę:

```
>>> t = tuple()
>>> print(t)
()
```

Jeśli argumentem jest sekwencja (ciąg znaków, lista lub krotka), to wynikiem wywołania funkcji `tuple()` jest krotka z elementami podanej sekwencji:

[1] Ciekawostka: Angielskie tłumaczenie słowa „krotka", czyli *tuple*, pochodzi od nazw nadawanych sekwencjom cyfr o różnej długości: *single, double, triple, quadruple, quintuple, sextuple, septuple* itd.

```
>>> t = tuple('łubin')
>>> print(t)
('ł', 'u', 'b', 'i', 'n')
```

Ponieważ słowo `tuple` jest nazwą konstruktora, powinieneś unikać używania go jako nazwy zmiennej.

Większość operatorów związanych z listami działa również na krotkach. Operator nawiasów kwadratowych służy do podawania indeksu (pozycji) elementu:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print(t[0])
'a'
```

Operator wycinania wybiera szereg elementów.

```
>>> print(t[1:3])
('b', 'c')
```

Ale jeśli spróbujesz zmodyfikować jeden z elementów krotki, to otrzymasz błąd:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

Nie możesz modyfikować elementów krotki, ale możesz zastąpić jedną krotkę inną krotką:

```
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```

10.2. Porównywanie krotek

Operatory porównania działają z krotkami i innymi sekwencjami. Python zaczyna od porównania pierwszego elementu z każdej sekwencji. Jeśli są one równe, przechodzi do następnego elementu itd., aż znajdzie takie elementy, które się różnią. Kolejne elementy nie są już brane pod uwagę (nawet jeśli ich wartość jest bardzo duża).

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

Funkcja `sort()` działa w ten sam sposób. Sortuje przede wszystkim według pierwszego elementu, ale w przypadku takich samych wartości sortuje według drugiego elementu itd.

Powyższa własność prowadzi do schematu zwanego *DSU* oznaczającego

Dekorowanie sekwencji poprzez zbudowanie listy krotek z jednym lub wieloma kluczami sortowania, które poprzedzają elementy sekwencji,
Sortowanie listy krotek przy pomocy funkcji wbudowanej `sort()`,
Usunięcie dekorowania poprzez wyodrębnienie posortowanych elementów sekwencji.

Załóżmy, że masz listę wyrazów i chcesz je posortować od najdłuższego do najkrótszego:

```
txt = 'but soft what light in yonder window breaks'
words = txt.split()
t = list()
for word in words:
    t.append((len(word), word))

t.sort(reverse=True)

res = list()
for length, word in t:
    res.append(word)

print(res)

# Kod źródłowy: https://py4e.pl/code3/soft.py
```

Pierwsza pętla buduje listę krotek, gdzie każda krotka jest wyrazem poprzedzonym jego długością.

Funkcja `sort()` porównuje najpierw pierwsze elementy krotek, czyli długości wyrazów, a drugie elementy rozpatruje tylko w przypadku takich samych długości wyrazów. Argument z podaną nazwą `reverse=True` wskazuje funkcji `sort()`, by sortować w kolejności malejącej.

Druga pętla przechodzi przez listę krotek i tworzy listę wyrazów według ich malejącej długości. Czteroznakowe wyrazy są posortowane w odwrotnej kolejności alfabetycznej, więc słowo „what" pojawia się przed słowem „soft".

Wynik działania programu jest następujący:

```
['yonder', 'window', 'breaks', 'light', 'what', 'soft', 'but', 'in']
```

Oczywiście linia ta, gdy zostanie przekształcona w listę Pythona i posortowana w porządku malejącym według długości słów, traci swój poetycki urok.

10.3. Krotki i operacja przypisania

Jedną z unikalnych cech syntaktycznych języka Python jest możliwość posiadania krotki po lewej stronie instrukcji przypisania. Pozwala to na przypisanie w tym samym czasie więcej niż jednej zmiennej, w momencie gdy lewa strona jest sekwencją.

W poniższym przykładzie mamy dwuelementową listę (która jest sekwencją) i w jednej instrukcji przypisujemy pierwszy i drugi element sekwencji do zmiennych `x` i `y`.

```
>>> m = [ 'miłej', 'zabawy' ]
>>> x, y = m
>>> x
'miłej'
>>> y
'zabawy'
>>>
```

To nie jest magia, Python *z grubsza* tłumaczy składnię przypisania krotek na następującą:[2].

```
>>> m = [ 'miłej', 'zabawy' ]
>>> x = m[0]
>>> y = m[1]
```

[2]Python nie tłumaczy składni dosłownie. Na przykład, jeśli spróbujesz takiej operacji na słownikach, to nie będzie to działać tak, jak można by się tego spodziewać

```
>>> x
'miłej'
>>> y
'zabawy'
>>>
```

Stylistycznie rzecz ujmując, gdy używamy krotki po lewej stronie instrukcji przypisania, pomijamy nawiasy, ale poniższa składnia jest również poprawna:

```
>>> m = [ 'miłej', 'zabawy' ]
>>> (x, y) = m
>>> x
'miłej'
>>> y
'zabawy'
>>>
```

Sprytne zastosowanie operacji przypisania krotki pozwala nam na *zamianę* wartości dwóch zmiennych przy pomocy jednej instrukcji:

```
>>> a, b = b, a
```

Obie strony tego wyrażenia są krotkami, ale lewa strona jest krotką zmiennych; prawa strona jest krotką wyrażeń. Każda wartość po prawej stronie jest przypisana do odpowiadającej jej zmiennej po lewej stronie. Wszystkie wyrażenia po prawej stronie są ewaluowane przed każdym z przypisań.

Liczba zmiennych po lewej stronie i liczba wartości po prawej stronie muszą być takie same:

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

Ogólnie rzecz biorąc, prawa strona może być dowolną sekwencją (ciągiem, listą lub krotką). Na przykład, aby podzielić adres e-mail na nazwę użytkownika i domenę, można napisać poniższy kod:

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

Wartość zwracana przez `split()` jest listą z dwoma elementami; pierwszy element jest przypisany do `uname`, a drugi – do `domain`.

```
>>> print(uname)
monty
>>> print(domain)
python.org
```

10.4. Słowniki i krotki

Słowniki posiadają metodę o nazwie `items()`, która zwraca zbiór krotek, gdzie każda krotka jest parą klucz-wartość. Domyślnie zwracany przez tę metodę obiekt jest typu `dict_items`, ale możemy go spokojnie zamienić na listę, dzięki czemu będziemy mieli dostęp do metod związanych z listami:

```
>>> d = {'c': 22, 'a': 10, 'b': 1}
>>> t = list(d.items())
>>> print(t)
[('c', 22), ('a', 10), ('b', 1)]
```

Ponieważ jednak lista krotek jest listą, a krotki są porównywalne, to możemy teraz posortować listę krotek. Konwertowanie słownika na listę krotek jest dla nas sposobem na wyciągnięcie zawartości słownika posortowanego po kluczach:

```
>>> t.sort()
>>> print(t)
[('a', 10), ('b', 1), ('c', 22)]
```

Nowa lista jest posortowana rosnąco w porządku alfabetycznym według kluczy.

10.5. Wielokrotne przypisanie w słownikach

Używając razem `items()`, przypisania krotki oraz `for`, możesz zobaczyć ciekawy schemat kodu do przechodzenia po kluczach i wartościach słownika w jednej pętli:

```
d = {'c': 22, 'a': 10, 'b': 1}
for key, val in d.items():
    print(val, key)
```

Powyższa pętla ma dwie *zmienne iteracyjne*, ponieważ `d.items()` zwraca zbiór krotek, a `key, val` jest przypisaniem krotki, które kolejno iteruje przez każdą z par klucz-wartość w słowniku.

W każdej kolejnej iteracji pętli zarówno `key`, jak i `val` posuwają się naprzód do kolejnej pary klucz-wartość w słowniku.

Wynik tej pętli jest następujący:

```
22 c
10 a
1 b
```

Jeżeli połączymy te dwie techniki, możemy wypisać zawartość słownika posortowaną po *wartościach* zapisanych w każdej parze klucz-wartość.

Aby to zrobić, najpierw sporządzamy listę krotek, gdzie każda krotka to `(value, key)`. Metoda `items()` zwróci nam listę `(key, value)` krotek, ale tym razem chcemy sortować według wartości, a nie klucza. Po skonstruowaniu listy z parami klucz-wartość, możemy łatwo posortować listę w odwrotnej kolejności i wypisać nową, posortowaną listę.

```
>>> d = {'a': 10, 'c': 22, 'b': 1}
>>> t = list()
>>> for key, val in d.items():
...     t.append( (val, key) )
...
>>> t
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> t.sort(reverse=True)
>>> t
[(22, 'c'), (10, 'a'), (1, 'b')]
>>>
```

Ostrożnie konstruując listę krotek tak, aby w każdej z nich wartość była pierwszym elementem, możemy posortować listę krotek i uzyskać zawartość słownika posortowaną po wartościach.

10.6. Najczęściej występujące słowa

Wracając do naszego przykładu z tekstem z *Romeo i Julia* (Akt II, Scena II), możemy rozszerzyć nasz program po to, by użyć opisanej wyżej techniki do wypisania dziesięciu najczęściej używanych słów w tekście:

```
import string
fhand = open('romeo-full.txt')
counts = dict()
for line in fhand:
    line = line.translate(str.maketrans('', '', string.punctuation))
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

# Posortuj słownik według wartości
lst = list()
for key, val in list(counts.items()):
    lst.append((val, key))

lst.sort(reverse=True)

for key, val in lst[:10]:
    print(key, val)

# Kod źródłowy: https://py4e.pl/code3/count3.py
```

Pierwsza część programu, która odczytuje plik i tworzy słownik, który mapuje każde słowo do liczby jego wystąpień w dokumencie, pozostaje niezmieniona. Ale zamiast po prostu wypisać `counts` i zakończyć program, konstruujemy listę krotek `(val, key)`, a następnie sortujemy tę listę w odwrotnej kolejności.

Ponieważ wartość jest pierwsza, będzie ona użyta do porównań. Jeśli jest więcej niż jedna krotka o tej samej wartości, to operacja sortowania spojrzy na drugi element (klucz), więc krotki, w których wartość jest taka sama, będą posortowane w kolejności alfabetycznej po kluczu.

Na koniec piszemy ładną pętlę `for`, która podczas iteracji wykonuje operację przypisania z fragmentu listy (`lst[:10]`) i wypisuje dziesięć najczęściej występujących słów.

Tak więc teraz wynik programu w końcu wygląda tak, jak tego oczekujemy od naszej analizy częstości słów.

```
61 i
42 and
40 romeo
34 to
34 the
32 thou
32 juliet
30 that
29 my
24 thee
```

Fakt, że takie skomplikowane parsowanie i analiza danych mogą być wykonane za pomocą łatwego do zrozumienia 19-wierszowego programu Pythona, jest jednym z powodów, dla których Python jako język do eksploracji informacji to dobry wybór.

10.7. Listy składane

W poprzednim przykładzie utworzyliśmy listę przy pomocy poniższego kodu:

```
lst = list()
for key, val in list(counts.items()):
    lst.append((val, key))
```

Python posiada możliwość tworzenia dynamicznych list, które w pewnym sensie zawierają pętlę `for` wewnątrz jej definicji. Mechanizm ten nazywamy *listą składaną* (ang. *list comprehension*) i uzyskujemy dzięki niemu bardziej kompaktowy kod.

W poprzednim przykładzie tworzenie listy moglibyśmy zapisać następująco:

```
lst = [ (v,k) for k,v in counts.items() ]
```

Teraz kod naszego programu mógłby wyglądać tak:

```
import string
fhand = open('romeo-full.txt')
counts = dict()
for line in fhand:
    line = line.translate(str.maketrans('', '', string.punctuation))
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

# Posortuj słownik według wartości
lst = [(v, k) for k, v in counts.items()]

lst.sort(reverse=True)

for key, val in lst[:10]:
    print(key, val)

# Kod źródłowy: https://py4e.pl/code3/count4.py
```

10.8. Używanie krotek jako kluczy w słownikach

Ponieważ krotki są *haszowalne*, a listy nie są, to jeśli chcemy utworzyć *złożony* klucz do użycia w słowniku, musimy użyć krotki jako klucza.

Napotkalibyśmy klucz złożony, gdybyśmy chcieli stworzyć książkę telefoniczną, która mapuje z nazwisko-imię do numerów telefonów. Zakładając, że zdefiniowaliśmy zmienne `last`, `first` i `number`, moglibyśmy napisać instrukcję przypisania do słownika w następujący sposób:

```
directory[last,first] = number
```

Wyrażenie w nawiasach jest krotką. Moglibyśmy użyć przypisania krotki w pętli `for`, tak aby przejść po takim słowniku.

```
for last, first in directory:
    print(first, last, directory[last,first])
```

Powyższa pętla przechodzi przez klucze w `directory`, które są krotkami. Przypisuje elementy każdej krotki do `last` i `first`, a następnie wypisuje nazwę i odpowiadający jej numer telefonu.

10.9. Sekwencje: ciągi znaków, listy i krotki – O rany!

Skupiłem się na listach krotek, ale prawie wszystkie przykłady w tym rozdziale działają również z listami list, krotkami krotek i krotkami list. Aby uniknąć wyliczania możliwych kombinacji, czasem łatwiej jest mówić o sekwencjach sekwencji.

W wielu kontekstach różne rodzaje sekwencji (ciągów znaków, list i krotek) mogą być używane zamiennie. Więc jak i dlaczego wybierasz jedną z nich zamiast innych?

Zaczynając od tego, co oczywiste, ciągi znaków są bardziej ograniczone niż inne sekwencje, ponieważ ich elementy muszą być znakami. Są one również niezmienne. Jeśli potrzebujesz możliwości zmiany znaków w takim ciągu (w przeciwieństwie do tworzenia nowego ciągu), możesz w zamian użyć listy znaków.

Listy są popularniejsze w użyciu niż krotki, głównie dlatego, że są zmienne. Jednak jest kilka przypadków, w których możesz preferować krotki:

1. W niektórych kontekstach, jak np. instrukcja `return`, stworzenie krotki jest składniowo prostsze od stworzenia listy. W innych przepadkach możesz preferować listę.
2. Jeśli chcesz użyć sekwencji jako klucza w słowniku, musisz użyć typu niezmiennego, takiego jak krotka lub ciąg znaków.
3. Jeżeli przekazujesz sekwencję jako argument do funkcji, używanie krotek zmniejsza możliwość nieoczekiwanego działania kodu spowodowanego aliasowaniem.

Ponieważ krotki są niezmienne, nie dostarczają metod takich jak `sort()` i `reverse()`, które modyfikują istniejące listy. Jednakże Python dostarcza wbudowane funkcje `sorted()` i `reversed()`, które przyjmują każdą sekwencję jako parametr i zwracają nową sekwencję z tymi samymi elementami w innej kolejności.

10.10. Debugowanie

Listy, słowniki i krotki są ogólnie znane jako *struktury danych*; w tym rozdziale zaczęliśmy zauważać złożone struktury danych, takie jak listy krotek oraz słowniki, które zawierają krotki jako klucze i listy jako wartości. Złożone struktury danych są użyteczne, ale są podatne na to, co nazywam *błędami kształtu*, tzn. błędy spowodowane niewłaściwym typem, rozmiarem lub zawartością struktury danych; być może piszesz też jakiś kod i zapominasz o kształcie swoich danych, przez co nieświadomie umieszczasz błąd w swoim programie. Na przykład, jeśli oczekujesz listy składającej się z jednej liczby całkowitej, a ja daję Ci po prostu zwykłą liczbę całkowitą (która nie jest opakowana listą), Twój kod nie będzie działać.

10.11. Słowniczek

DSU Skrót od „dekorowanie-sortowanie-usunięcie dekorowania"; schemat, który polega na budowaniu listy krotek, sortowaniu i wyodrębnianiu części wyniku.

haszowalny Typ, który posiada funkcję haszowania. Typy niezmienne, takie jak liczby całkowite, zmiennoprzecinkowe i ciągi znaków, są haszowalne; typy zmienne, takie jak listy i słowniki, nie są haszowalne.

krotka Niezmienna sekwencja elementów.

kształt (struktury danych) Podsumowanie typu, rozmiaru i zawartości struktury danych.

porównywalny Typ, w którym można sprawdzić, czy jedna wartość jest większa, mniejsza lub równa innej wartości tego samego typu. Typy, które są porównywalne, można umieścić na liście i posortować.

przypisanie do krotki Przypisanie z sekwencją po prawej stronie i krotką zmiennych po lewej. Prawa strona jest ewaluowana, a następnie jej elementy są przypisywane do zmiennych po lewej stronie.

struktura danych Kolekcja powiązanych ze sobą wartości, często zorganizowanych w listy, słowniki, krotki itp.

10.12. Ćwiczenia

Ćwiczenie 1. Przepisz swój poprzedni program zliczający wysłane wiadomości mailowe w następujący sposób. Wczytaj i przetwórz linie zaczynające się od „From" oraz wyciągnij z nich adresy mailowe. Za pomocą słownika dla każdej osoby zapisz liczbę wysłanych przez nią wiadomości.

Po przeczytaniu wszystkich danych utwórz ze słownika listę krotek w postaci *(liczba, adres e-mail)*. Następnie posortuj listę w odwrotnej kolejności i wypisz osobę, która wysłała najwięcej wiadomości.

Przykładowa linia:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Przykładowe uruchomienia:

```
Podaj nazwę pliku: mbox-short.txt
cwen@iupui.edu 5

Podaj nazwę pliku: mbox.txt
zqian@umich.edu 195
```

Ćwiczenie 2. Napisz program, który zlicza rozkład godzin dla wszystkich wiadomości. Możesz wyciągnąć godzinę z wiersza „From", odszukując ciąg znaków związany z czasem, a następnie dzieląc ten ciąg na części za pomocą znaku dwukropka. Kiedy już zgromadzisz wartości dla każdej godziny, wypisz zliczenia, po jednym na wiersz, posortowane według godzin, tak jak pokazano poniżej.

```
Podaj nazwę pliku: mbox-short.txt
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```

Ćwiczenie 3. Napisz program, który wczytuje plik i wypisuje *litery* malejąco po częstości ich występowania. Twój program powinien przekonwertować wszystkie dane wejściowe na małe litery i zliczać tylko litery a-z. Program nie powinien uwzględniać spacji, cyfr, interpunkcji ani niczego innego poza literami a-z. Znajdź próbki tekstów z kilku różnych języków i sprawdź, jak bardzo częstość występowania liter różni się w zależności od języka. Porównaj swoje wyniki z tabelami pod adresem https://en.wikipedia.org/wiki/Letter_frequency.