SŁOWNIKI W PYTHONIE

Wszystkie przykłady zostały sprawdzone na Pythonie 2.5.1

Stringi, listy, krotki (tuples zwane również zbiorami) oraz **słowniki** zaliczają się do typów obiektów zwanych kontenerami i kolekcjami. Jednakże słowniki różnią się od pozostałych znacznie.
Czym jest słownik - słownik jest zmienny (modyfikowalny), nieposortowany zestaw par **klucz:wartość**. Wartością może być dowolny obiekt Pythonowy. Słownik można porównać do tablic asocjacyjnych z innych języków. Według książki "Learning Python" napisanej przez Marka Lutza, Davida Aschera wewnętrznie są implementowane jako tablice haszujące i dodatkowo optymalizowane, co powoduje bardzo szybkie wydobywanie danych. W przeciwieństwie do stringów, list, krotek słownik nie jest sekwencją gdyż w sekwencji istnieje porządek, słownik nie posiada porządku. Indeksem słownika jest klucz. Kluczem, według Python Tutorial, może być dowolny niezmienny typ. Liczby i stringi są niezmiennymi typami więc mogą stanowić klucz w słowniku. Krotka może stanowić klucz wyłącznie jeśli każdy jej element jest typem niezmiennym. Lista za to nie może stanowić klucza w słowniku ponieważ jest typem zmiennym. Słownik tworzymy "wkładając" pary klucz:wartość w nawiasy klamrowe, oddzielając je przecinkiem:

```
Dict = {klucz1:wartość1, klucz2:wartooość2, klucz3:wartooość3}
```

lub stworzyć pusty słownik:

```
Dict = {}
```

do pustego słownika można dodać elementy jak do tablicy:

```
Dict[index5]=wartość5
```

Rzeczą oczywistą jest, że klucze muszą być unikatowe. W przeciwnym razie - dodając parę klucz:war, gdzie klucz już istnieje w słowniku - zamażemy starą wartość przez war - chyba, że mamy taki zamiar by zmodyfikować starą wartość. Możliwe jest usuwanie elementu ze słownika przez słowo kluczowe **del**:

```
del Dict[index5]
```

Słowniki razem z listami i krotkami mają najwyższy priorytet operacji. Aby zobaczyć wszystkie możliwe do wykonania operacje na słowniku wywołamy:

```
slownik={}
dir(slownik)
```

W wyniku dostaniemy:

```
['clear', 'copy', 'fromkeys', 'get', 'has_key', 'items', 'iteritems', 'iterkeys',
'itervalues', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

W miejscu operacjami wykonywanymi na słowniku pojawiają się atrybuty.

clear - użycie: **Dict.clear()** nie zwraca wartości, usuwa wszystkie pary klucz:wartość ze słownika Dict.

copy - użycie: **Dict.copy()** zwraca Dict, płytkie kopiowanie słownika.

fromkeys - użycie: **Dict.fromkeys(S[,v])** - parametr v jest opcjonalny - tworzy słownik Dict z kluczami z S i wartościami równymi v.

get - użycie: **Dict.get(k[,d])** - parametr d jest opcjonalny - zwraca Dict[k] jeśli k jest indeksem w słowniku, w przeciwnym razie zwraca d.

has_key - użycie: **Dict.has_key(klucz1)** zwraca prawdę jeśli klucz1 należy do słownika Dict, w przeciwnym wypadku zwraca fałsz.

items - użycie: **Dict.items()** zwraca listę elementów (klucz, wartość).

iteritems - użycie: **Dict.iteritems()** iterator nad elementami słownika Dict.

iterkeys - użycie: **Dict.iterkeys()** iterator nad kluczami słownika Dict.

itervalues - użycie: **Dict.itervalues()** iterator nad wartościami słownika Dict.

keys - użycie: **Dict.keys()** zwraca listę kluczy.

pop - użycie: **Dict.pop(klucz[,blad])** - parametr błąd jest opcjonalny - zwraca wartość elementu korespondującego z kluczem i usuwa go ze słownika, jeśli klucza nie ma w słowniku zwraca wartość błąd, a jeśli nie została podana "zwraca" błąd - KeyError.

popitem - użycie: **Dict.popitem()** zwraca parę (klucz, wartość) i usuwa ze słownika, a jeśli słownik jest pusty "zwraca" błąd- KeyError.

setdefault - użycie: **Dict.setdefault(klucz,błąd)** - parametr błąd jest opcjonalny - zwraca wartość elementu korespondującego z kluczem, jeśli klucza nie ma w słowniku zwraca wartość błąd i wstawia do słownika parę klucz:błąd.

update - użycie: **Dict.update(E[,**F])** nie zwraca wartości, aktualizuje Dict parami klucz:wartość ze słowników podanych jako parametry.

values - użycie: **Dict.values()** zwraca listę wartości słownika.

Przykład:

```
#!/usr/bin/env python
# -*- coding: iso-8859-2 -*-
# z pliku wczytujemy dane nazwa email
# i wpisujemy do słownika
def wyslij_do(do,tresc):
    print "wysłałeś maila do ",
    print do,
    print "o treści: ",
    print tresc
mail={}
tresc="Python jest super"
infile = file('t.dat', 'r')
# można by wczytywać z bazy danych używając np.: MySQLdb
for line in infile:
    line=line.strip('\n').split(' ')
    mail[line[0]]=line[1]
print mail.keys()
print len(mail)
for o in mail.itervalues():
    wyslij_do(o,tresc)
#Tak mógłby powstać interfejs www do odbierania/wysyłania poczty
```

Przykład użycia słownika jako kontenera do przechowywania danych programu oraz jego opcji wraz z reakcją na nie. Kod użyczony przez Marcina "KoD" Walkowiaka w celu prezentacji możliwości słownika.

```
_cmdLineConfig = {
"usage": "%prog {[options] --address=ADDRESS | --config-file=CONFIGFILE}",
"version": "%prog 1.0",
"description": "",
"option_list":
[make_option("--address",action = "store",type = string",dest = "address",help = "url to web page or site to download")
]
}
```

Podsumowanie!

Słowniki mają bardzo szerokie zastosowanie chociażby od takich jak książka telefoniczna, kalendarz urodzinowy, po wczytywanie z bazy danych i wyświetlanie.

Autor: slyher