

Za radoznale: Kolekcije

Ovaj fajl je za one koji žele više. Ništa oдавde nije potrebno za praćenje časova. Svaki deo počinje primerom koji smo radili na času, a zatim pokazuje kako se ista stvar može zapisati kraće ili naprednije. Svi primeri su i u folderu primeri/za_radoznale, pa ih možete pokrenuti, menjati i gledati šta se dešava.

1. itertools: sve kombinacije boja

Na času smo skupom pronašli koje se boje pojavljuju na slici, svaka samo jednom:

```
pikseli = ["crna", "bela", "crna", "plava", "crna", "bela"]
boje = set(pikseli)
print(sorted(boje))
```

Ispis:

```
['bela', 'crna', 'plava']
```

Recimo da od nekoliko boja želimo da isprobamo sve dvobojne kombinacije za poster. Modul itertools ima funkciju combinations, koja daje sve izbore od dva elementa, a svaki izbor je torka. Isti par se ne ponavlja obrnutim redom, jer je (crna, plava) za nas isto što i (plava, crna).

```
from itertools import combinations

boje = ["bela", "crna", "plava", "žuta"]
for par in combinations(boje, 2):
    print(par)

print("Trobojnih kombinacija:", len(list(combinations(boje, 3))))
```

Ispis:

```
('bela', 'crna')
('bela', 'plava')
('bela', 'žuta')
('crna', 'plava')
('crna', 'žuta')
('plava', 'žuta')
Trobojnih kombinacija: 4
```

U starim beleškama postojao je i ovaj primer, koji pravi sve podskupove zadate veličine. Funkcija podskupovi od svake torke koju da combinations pravi skup. Sve to radi u jednom redu, pomoću list comprehension:

```
import itertools as it

def podskupovi(skup, velicina_podskupa):
    return [set(i) for i in it.combinations(skup, velicina_podskupa)]

A = {1, 2, 3, 4, 5}
n = 4

print(podskupovi(A, n))
```

Ispis:

```
[{1, 2, 3, 4}, {1, 2, 3, 5}, {1, 2, 4, 5}, {1, 3, 4, 5}, {2, 3, 4, 5}]
```

Zašto ovo nismo radili na času: Ovaj primer odjednom koristi uvoz modula, pisanje funkcije i list comprehension. Module i funkcije radimo tek u sledećoj temi, pa bi na času o kolekcijama to bilo previše novih stvari odjednom.

2. Counter: brojač koji već postoji

Na vežbi (Level 3) smo napravili brojač boja pomoću rečnika i metode get:

```
pikseli = ["crna", "bela", "crna", "plava", "crna", "bela"]
broj = {}
for boja in pikseli:
    broj[boja] = broj.get(boja, 0) + 1
print(broj)
```

Ispis:

```
{'crna': 3, 'bela': 2, 'plava': 1}
```

Brojanje je toliko često da Python za njega ima gotovu kolekciju, Counter, u modulu collections. Counter se ponaša kao rečnik u kom je vrednost broj pojavljivanja, a metoda most_common odmah daje najčešće elemente.

```
from collections import Counter

pikseli = ["crna", "bela", "crna", "plava", "crna", "bela"]
broj = Counter(pikseli)
print(broj)
print(broj["crna"])
print(broj.most_common(1))
```

Ispis:

```
Counter({'crna': 3, 'bela': 2, 'plava': 1})
3
[('crna', 3)]
```

most_common(1) vraća listu sa jednim parom (boja, broj), onim koji se pojavljuje najviše puta.

Zašto ovo nismo radili na času: Hteli smo da sami napišemo brojač i da vidimo kako get(boja, 0) + 1 radi korak po korak. Kad to razumemo, gotov Counter više nije crna kutija.

3. Rečnik u jednom redu

Na času smo prolazili kroz parove ključ-vrednost metodom items():

```
paleta = {"crvena": (230, 57, 70), "teget": (29, 53, 87), "krem": (241, 250, 238)}
for ime, boja in paleta.items():
    print(ime, sum(boja))
```

Ispis:

```
crvena 357
teget 169
krem 729
```

Kao što list comprehension pravi listu u jednom redu, tako i rečnik može da se napravi u jednom redu, u vitičastim zagradama i sa dvotačkom između ključa i vrednosti. Prvi rečnik čuva svetlinu svake boje, a drugi zadržava samo svetle boje.

```
paleta = {"crvena": (230, 57, 70), "teget": (29, 53, 87), "krem": (241, 250, 238)}

svetlina = {ime: sum(boja) for ime, boja in paleta.items()}
print(svetlina)

svetle = {ime: boja for ime, boja in paleta.items() if sum(boja) > 400}
print(svetle)
```

Ispis:

```
{'crvena': 357, 'teget': 169, 'krem': 729}
{'krem': (241, 250, 238)}
```

Zašto ovo nismo radili na času: List comprehension smo upoznali tek u prethodnoj temi. Na času je bilo važnije da dobro uvežbamo items() u običnoj petlji, koja se lakše čita.

4. Još operacija sa skupovima

Na času smo radili uniju, presek i razliku dva skupa boja:

```
slika_a = {"crvena", "plava", "žuta"}
slika_b = {"plava", "zelena"}
print(sorted(slika_a | slika_b))
print(sorted(slika_a & slika_b))
print(sorted(slika_a - slika_b))
```

Ispis:

```
['crvena', 'plava', 'zelena', 'žuta']
['plava']
['crvena', 'žuta']
```

Operator ^ daje boje koje su samo na jednoj od dve slike, a ne na obe; to se zove simetrična razlika. Operator <= proverava da li je ceo prvi skup sadržan u drugom. Na kraju, skup može da se napravi i u jednom redu, kao lista: ovde su to sve različite dužine reči.

```
slika_a = {"crvena", "plava", "žuta"}
slika_b = {"plava", "zelena"}
print(sorted(slika_a ^ slika_b))
print({"plava"} <= slika_a, slika_b <= slika_a)

reci = ["kist", "boja", "platno", "ram"]
duzine = {len(r) for r in reci}
print(sorted(duzine))
```

Ispis:

```
['crvena', 'zelena', 'žuta']
True False
[3, 4, 6]
```

Zašto ovo nismo radili na času: Unija, presek i razlika pokrivaju gotovo sve što nam u igrama treba. Ostale operacije lako se nađu u dokumentaciji kad zatrebaju.

5. Torka kao ključ rečnika: mapa sveta

U primeru sa komandama sa tastature torke su bile vrednosti u rečniku:

```
komande = {"w": (0, -1), "s": (0, 1), "a": (-1, 0), "d": (1, 0)}
dx, dy = komande["d"]
print(dx, dy)
```

Ispis:

```
1 0
```

Torka može da bude i ključ, jer se ne menja. Tako dobijamo mapu sveta: za svako polje (x, y) pamtimmo šta se na njemu nalazi. Za polja koja nismo upisali get vraća „trava“, pa ne moramo da upisujemo celu mapu.

```
mapa = {(0, 0): "kuća", (1, 0): "reka", (1, 1): "most"}
print(mapa[(1, 0)])

lik = (0, 0)
for korak in [(1, 0), (0, 1), (1, 0)]:
    lik = (lik[0] + korak[0], lik[1] + korak[1])
    print(lik, mapa.get(lik, "trava"))
```

Ispis:

```
reka
(1, 0) reka
(1, 1) most
(2, 1) trava
```

Lista ne bi mogla da bude ključ, jer se menja. Zato je za koordinate torka pravi izbor.

Zašto ovo nismo radili na času: Na času smo rečnik upoznali sa stringovima kao ključevima, što je najčešći slučaj. Torke kao ključevi postaju korisne tek kad pravimo veće igre sa mapom.