

Za radoznale: Liste

Ovaj fajl je za one koji žele više. Ništa odavde 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. lambda: mala funkcija za argument key

Na času smo paletu sortirali po svetlini tako što smo za key dali funkciju sum, koja sabira tri broja u boji:

```
paleta = [(230, 57, 70), (29, 53, 87), (241, 250, 238), (168, 218, 220), (69, 123, 157)]
print(sorted(paleta, key=sum))
```

Ispis:

```
[(29, 53, 87), (69, 123, 157), (230, 57, 70), (168, 218, 220), (241, 250, 238)]
```

A šta ako želimo da sortiramo po nečemu za šta ne postoji gotova funkcija, na primer samo po plavoj komponenti? Tada funkciju napravimo na licu mesta, rečju lambda. Zapis lambda b: b[2] čita se „za boju b vrati njen treći broj“, a to je plava komponenta.

```
paleta = [(230, 57, 70), (29, 53, 87), (241, 250, 238), (168, 218, 220), (69, 123, 157)]
# po plavoj komponenti (indeks 2)
print(sorted(paleta, key=lambda b: b[2]))

# po svetlini kako je vidi oko: zelena se računa najviše, plava najmanje
print(sorted(paleta, key=lambda b: 0.3 * b[0] + 0.59 * b[1] + 0.11 * b[2]))
```

Ispis:

```
[(230, 57, 70), (29, 53, 87), (69, 123, 157), (168, 218, 220), (241, 250, 238)]
[(29, 53, 87), (230, 57, 70), (69, 123, 157), (168, 218, 220), (241, 250, 238)]
```

Druga lambda računa svetlinu onako kako to rade programi za obradu slike, jer je oko osetljivije na zelenu nego na plavu. Uporedite poslednji ispis sa sortiranjem po sum: crvena (230, 57, 70) i plava (69, 123, 157) zamenile su mesta, jer oko ovu plavu vidi kao svetliju nego što kaže običan zbir.

Zašto ovo nismo radili na času: Reč lambda pravi funkciju, a funkcije pišemo tek u temi 11. Za svetlinu nam je bio dovoljan sum, koji smo već znali, pa nije bilo potrebe za novom rečju.

2. enumerate: indeks i element odjednom

Na času smo, kad su nam trebali i indeks i element, pisali petlju kroz indekse:

```
boje = ["oker", "teget", "bordo"]
for i in range(len(boje)):
    print(i, boje[i])
```

Ispis:

```
0 oker
1 teget
2 bordo
```

Funkcija `enumerate` u svakom krugu petlje daje dve vrednosti: redni broj i element. Zato iza `for` pišemo dva imena. Sa `start=1` brojanje počinje od jedan, što je zgodno za spiskove koje čitaju ljudi.

```
boje = ["oker", "teget", "bordo"]
for i, boja in enumerate(boje):
    print(i, boja)

for broj, boja in enumerate(boje, start=1):
    print(f"{broj}. {boja}")
```

Ispis:

```
0 oker
1 teget
2 bordo
1. oker
2. teget
3. bordo
```

Zašto ovo nismo radili na času: Petlja sa dva imena iza `for` oslanja se na torke, koje radimo u sledećoj temi. Na času je bilo važnije da jasno vidimo indekse, a `range(len(boje))` ih pokazuje direktno.

3. zip: dve liste u paru

Kad imamo dve liste iste dužine, na primer imena boja i njihove brojeve, na času bismo ih prošli zajedničkim indeksom:

```
imena = ["crvena", "teget", "krem"]
paleta = [(230, 57, 70), (29, 53, 87), (241, 250, 238)]
for i in range(len(imena)):
    print(imena[i], paleta[i])
```

Ispis:

```
crvena (230, 57, 70)
teget (29, 53, 87)
krem (241, 250, 238)
```

Funkcija `zip` uzima prvi element iz obe liste, pa drugi iz obe i tako redom, kao rajsferšlus. Od takvih parova odmah možemo da napravimo i rečnik, u kom se boja traži po imenu.

```
imena = ["crvena", "teget", "krem"]
paleta = [(230, 57, 70), (29, 53, 87), (241, 250, 238)]
for ime, boja in zip(imena, paleta):
    print(ime, boja)

po_imenu = dict(zip(imena, paleta))
print(po_imenu["teget"])
```

Ispis:

```
crvena (230, 57, 70)
teget (29, 53, 87)
krem (241, 250, 238)
(29, 53, 87)
```

Zašto ovo nismo radili na času: zip daje parove kao torke, a rečnik pravimo tek u sledećoj temi. Bez te dve stvari zip bi izgledao kao čarolija.

4. list comprehension sa uslovom i sa izborom

Na času smo napravili listu parnih brojeva, sa uslovom na kraju:

```
parni = [x for x in range(10) if x % 2 == 0]
print(parni)
```

Ispis:

```
[0, 2, 4, 6, 8]
```

Isti oblik radi i sa paletom. Prvi red bira samo svetle boje, drugi pravi negativ svake boje, a treći svakoj boji daje oznaku. U trećem redu if i else stoje ispred for i biraju jednu od dve vrednosti, a ne izbacuju elemente.

```
paleta = [(230, 57, 70), (29, 53, 87), (241, 250, 238), (168, 218, 220), (69, 123, 157)]
svetle = [b for b in paleta if sum(b) > 400]
print(svetle)

negativ = [(255 - r, 255 - g, 255 - b) for (r, g, b) in paleta]
print(negativ)

oznake = ["svetla" if sum(b) > 400 else "tamna" for b in paleta]
print(oznake)
```

Ispis:

```
[(241, 250, 238), (168, 218, 220)]
[(25, 198, 185), (226, 202, 168), (14, 5, 17), (87, 37, 35), (186, 132, 98)]
['tamna', 'tamna', 'svetla', 'svetla', 'tamna']
```

Zašto ovo nismo radili na času: Kad se u jednom redu nađu i uslov na kraju i izbor if-else na početku, lako ih je pomešati. Na času smo hteli da list comprehension ostane kratak i čitljiv.

5. Lista lista: mreža piksela

Na času smo videli da množenje liste brojem ponavlja njene elemente:

```
nova_lista = [5] * 10
print(nova_lista)
```

Ispis:

```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
```

Slika je mreža piksela, pa je zgodno da je čuvamo kao listu redova, gde je svaki red lista. Element biramo sa dva indeksa: prvi je red, a drugi kolona. Pažnja: ako mrežu napravimo množenjem, dobijamo tri imena za

isti red, kao kod $b = a$.

```
mreza = [[0] * 4 for red in range(3)]
mreza[1][2] = 1
for red in mreza:
    print(red)

print("---")
losa = [[0] * 4] * 3
losa[1][2] = 1
for red in losa:
    print(red)
```

Ispis:

```
[0, 0, 0, 0]
[0, 0, 1, 0]
[0, 0, 0, 0]
---
[0, 0, 1, 0]
[0, 0, 1, 0]
[0, 0, 1, 0]
```

U drugoj mreži jedinica se pojavila u svakom redu, jer su sva tri reda ista lista. Prva mreža pravi novi red u svakom krugu, pa je jedinica samo tamo gde smo je stavili.

Zašto ovo nismo radili na času: Dva indeksa zaredom i problem kopiranja u istom primeru su previše novih stvari odjednom. Na času je bilo dovoljno da razumemo kopiranje jedne liste.