Skript jezici V04 – Naprednija sintaksa

Milan Tomić

Sadržaj

- *Comprehension*
- Generatori
- Iteratori
- Korisne ugrađene funkcije

Comprehension

- Jedan od načina da se definišu kolekcije je pomoću tzv. *comprehension* izraza
- U suštini, svaki *comprehension* je kombinacija nekog izraza i for petlje koja će taj izraz izračunati za svaki element for petlje
- Na for petlju može se nadovezati druga for petlja ili if grananje, pri čemu će se izraz za dodatu for petlju računati kao da je ugnježdena petlja, a za if samo ako je uslov ispunjen
- Vrednost svakog tako izračunatog izraza postaće element kolekcije (liste, n-torke, skupa, rečnika)

List comprehension

```
#%% List comprehesion
l1 = [a for a in range(5)]
print(l1)  # [0, 1, 2, 3, 4]
l2 = [a for a in range(10) if a % 3 == 0]
print(l2)  # [0, 3, 6, 9]
l3 = [(a, b) for a in range(3) for b in range(2)]
print(l3)  # [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1)]
```

Set comprehension, dict comprehension

```
#%% Set comprehension
sc = {a for a in 'abracadabra'}
print(sc)  # {'d', 'r', 'c', 'b', 'a'}

#%% Dict comprehension
dc = {a : a**3 for a in range(4)}
print(dc)  # {0: 0, 1: 1, 2: 8, 3: 27}
```

Generatori

- Generatori su posebna vrsta objekta kroz koje može da se iterira, pri čemu svaka iteracija proceduralno generiše neku vrednost
- Generatori mogu biti konačni ili beskonačni
- Konačni generatori mogu da se proslede u konstruktor kolekcije, pri čemu će kolekcija sadržati generisane elemente
- Generator se definiše kao funkcija, koja kada treba da vrati vrednost, to radi uz pomoć rezervisane reči **yield**
- Kontekst napravljenog generatora se čuva i pri sledećoj iteraciji on kreće da se izvršava od poslednjeg yield poziva
- Ako je generator konačan i nema više vrednosti koje treba generisati, dovoljno je pozvati return

Primer generatora za generisanje svakog drugog broja od m do n

```
#%% Generator svakog drugog broja od m do n
def svaki_drugi(m, n):
    a = m + 1
    while a <= n:
        yield a
        a += 2

# kreiranje liste
a = list(svaki_drugi(1, 10))
print(a)  # [2, 4, 6, 8, 10]
# iteracija
for a in svaki_drugi(2, 10):
    print(a**2, end=' ')  #  9 25 49 81
print()

# ne resetuje se
g = svaki_drugi(1, 10)
print(list(g))  # [2, 4, 6, 8, 10]
print(list(g))  # []
```

Prednosti generatora

- Generatori se koriste u mnogo konteksta kada treba simulirati iteraciju kroz kolekciju, ali nije neophodno postojanje same kolekcije
- Zgodni su iz razloga što imaju mali memorijski potpis
- Svaki podatak se dobija (izračunava) tek onda kada je stvarno potreban
  - Ako nije potreban, neće oduzeti ni vreme ni memorijski prostor
- Na taj način mnoge povratne vrednosti funkcija u standardnoj biblioteci Python-a, koje su inače vraćale kolekcije, sada vraćaju generatore, jer je često kroz njih potrebno samo iterirati
  - Lako se konvertuju u kolekcije ako je kolekcija zaista neophodna
- Mogu biti beskonačni, dok kolekcije to ne mogu biti

Iteratori

- Iteratori su objekti koji služe za iteraciju (uglavnom kroz kolekcije)
- Da bi objekat klase bio iterabilan, potrebno je implementirati metode __iter__(self) i __next__(self)
- Metoda __iter__(self) služi za kreiranje novog iteratora
- Metoda __next__(self) služi za generisanje sledeće vrednosti u iteraciji
- Obe metode se posredno mogu pozvati funkcijama iter(obj) i next(obj)
- Takođe, obe metode se posredno pozivaju kada se započinje iteracija kroz objekat (for i in obj)
- Generatori su u suštini brži način da se implementira samostalni iterator (bez potrebe pisanja __iter__ i __next__, koji se interno obezbeđuju)

Iteratori

- Iterator je objekat koji ima __next__ metodu
- Iterabilni objekat ima __iter__ metodu
- Nekad se objekti pišu tako da su sami sebi iteratori, tj. imaju i __iter__ i __next__, pri čemu __iter__ vraća self
- Kada __next__ metoda nema više objekata za vraćanje, baca izuzetak StopIteration
- Direktnom implementacijom iteratora se mogu napraviti generatori koji se mogu pozvati više puta na istoj instanci (kao što je to range, koji se svaki put resetuje)
  - Pomoću generatora je to nemoguće, jer kada jednom naiđe na return, nema povratka nazad

Primer iteratora za svaki drugi broj od m do n (upotrebljiv više puta)

```
#%% Iterator - generator svakog drugog broja od m do n
class svaki_drugi:
    def __init__(self, m, n):
        self.a = m - 1
        self.m = m
        self.n = n

    def __iter__(self):
        return self

    def __next__(self):
        self.a += 2
        if self.a <= self.n:
            return self.a
        else:
            self.a = self.m - 1
            raise StopIteration

g = svaki_drugi(1, 10)
print(list(g))  # [2, 4, 6, 8, 10]
print(list(g))  # [2, 4, 6, 8, 10]
```

Korisne ugrađene funkcije

- Python ima veliki set ugrađenih funkcija, koje se mogu pronaći na sledećoj adresi: https://docs.python.org/3/library/functions.html#built-in-functions
- Zapravo, većinu ovih funkcija možete koristiti prilično intuitivno
- Set ovih funkcija se vremenom proširio, i verovatno je da će se u novim verzijama jezika još širiti
- Ovde ćemo navesti neke često korišćene funkcije, a bacićemo i osvrt na neke ređe korišćene

Manipulacija nad brojevima

- abs(x) – vraća apsolutnu vrednost broja (može biti realan ili ceo); ako je prosleđeni broj kompleksan, vraća njegov modul
- bin(x) – vraća binarni zapis broja, sa prefiksom 0b, kao string
- chr(x) – vraća string u kome je karakter čiji je unicode kod x (inverz mu je ord(x))
- complex(r, i) – vraća kompleksni broj čiji je realni deo r, imaginarni deo i. Kompleksni brojevi se pišu u zapisu ’r+ij’ tj. 1j je oznaka za imaginarnu jedinicu
- hex(x) – vraća heksadekadni zapis broja, sa prefiksom 0x, kao string
- oct(x) – vraća oktalni zapis broja, sa prefiksom 0o, kao string

Manipulacija nad brojevima

- pow(x, y[, z]) – vraća x**y, po modulu z ako je prosleđeno (efikasnije nego x**y % z)
- round(n, ndigits) – zaokružuje n na najbliži broj sa ndigits decimala (0 ako nije prosleđeno). Ako su vrednosti ekvidistantne, preferira parnu vrednost (npr. 1.5 se zaokružuje na 2 iako je jednako daleko od1)

Manipulacija nad kolekcijama

- all(it) – vraća True ako su svi objekti kolekcije (iterabilnog objekta) True pri konverziji u bool
- any(it) – vraća True ako je bilo koji objekat iterabilnog objekta True pri konverziji u bool
- enumerate(it) – vraća enumerate objekat, koji generiše parove (redni broj, obj) za svaki obj u it. Može da mu se prosledi start kao argument, inače počinje brojanje od 0
- filter(func, it) – pravi iterator onih objekata iz it za koje je func(obj) logički tačno; ako je func==None, koristi se identička funkcija
- iter(it) – vraća iterator objekta it
- len(it) – vraća veličinu kolekcije it

Manipulacija nad kolekcijama

- map(func, it, ...) – vraća iterator povratnih vrednosti kada se funkcija func primeni na svaku vrednost iz it; ako ima više prosleđenih iterabilnih objekata, funkcija se primenjuje paralelno na sve (tj. svaki je odgovarajući pozicioni argument funkcije)
- max(it, key, default) – vraća maksimum iterabilnog objekta; key i default su imenovani argumenti
- min(it, key, default) – vraća minimum iterabilnog objekta, kao max
- next(it, default) – vraća sledeći u iteraciji kroz it, default ako je prosleđen i ako je iteracija iscrpljena, inače baca izuzetak StopIteration
- reversed(it) – vraća kolekciju uređenu unazad
- sorted(it, key, reverse) – vraća kolekciju sortiranu po prosleđenom ključu (ako je prosleđen), u prosleđenom poretku (ako je prosleđen)

Manipulacija nad kolekcijama

- sum(it, start) – vraća sumu elemenata u kolekciji; start ne sme da bude string
- zip(*its) – vraća iterator koji agregira elemente prosleđenih iterabilnih objekata u n-torke; staje kada se iscrpe elementi iz najkraćeg

Zadaci

- vezbe04_zadaci.pdf