Kapitola 10
Rozhodování a cykly

10.10 Cyklus `for`

Definice Generátorové funkce jako zdroje

Jako zdroj hodnot parametru cyklu `for` můžeme definovat speciální funkci označovanou jako *generátorová funkce* (*generator function*). Tu poznáte podle toho, že má ve svém těle výraz složený z klíčového slova `yield` následovaného výrazem.

Překladač zpracovává generátorové funkce jinak než běžné funkce. Zavolání generátorové funkce se nespustí její tělo, ale vytvoří se speciální objekt – instance třídy `generator` z modulu `builtins`. Ten pak slouží jako generátor hodnot. Kód, který jste definovali jako tělo generátorové funkce, se spustí teprve aktivací tohoto generátoru.

Generátor aktivujete zavoláním funkce `next()`, které předáte daný generátor jako argument. Při prvním zavolání se spustí kód dané funkce, který se vykonává až do okamžiku, kdy narazí na výraz `yield`. Poté se spočte jeho operand, který funkce `next()` předá volajícímu programu jako funkční hodnotu.

Příští volání funkce `next()` pokračuje ve vykonávání kódu od místa, kde naposledy skončilo, tj. od naposledy provedeného výrazu `yield` a provádí jej až do okamžiku, kdy znovu narazí na výraz `yield`, anebo když funkce skončí.

Po ukončení kódu vyhodí každé následné volání funkce `next()` s daným argumentem výjimku `StopIteration`.

Zadáte-li v hlavičce příkazu `for` jako zdroj nějaký generátor, překladač potřebná volání funkce `next()` doplní a obdržené hodnoty přiřazuje parametru cyklu. Výjimku vyhozenou po posledním volání zachytí a cyklus řádně ukončí.

Demonstrační příklad definice generátoru a jeho použití je v učebnici ve výpisu 10.17 na straně 177. V něm je na řádcích `1–9` definována funkce `gen1` předvádějící malou ukázku některých možností generátorových funkcí. Na řádku `11` je pak použita v cyklu jako zdroj hodnot parametru cyklu. Výsledky spuštění cyklu čekají na řádcích `12–17`.

Generátorové funkce většinou obsahují pouze jeden výraz `yield`, ale chtěl jsem vám ukázat, že tomu tak být nemusí. Ve funkci `gen1()` je tento výraz použit ob řádek.

To jen ukazuje, že pokud se vám to hodí, můžete ve své generátorové funkci použít těchto výrazů několik.

Současně si povšimněte, že generátorová funkce může mít parametry, které ovlivňují její činnost (to už není taková extravagance, setkáte se s tím relativně běžně).

Operátor `yield` jako výraz

Již jsem se několikrát zmínil o výrazu `yield`, ale prozatím jsem ještě neprozradil, co je jeho hodnotou. Tady přichází jedna zvláštnost" hodnotu tohoto výrazu můžeme nastavit z místa, odkud je kód generátoru volán. Stačí místo funkce `next()` zavolat metodu `ggg.send()`, kde identifikátor `ggg` zastupuje generátor, jehož metodu voláte. Této metodě předáte jako argument výraz, jehož hodnotu vrátí výraz `yield` umístěný v kódu generátoru. Zadávání hodnoty výrazu `yield` se používá většinou k přenastavení hodnot parametrů cyklu, ale není to povinnost.

Zde je ale třeba upozornit na jednu důležitou skutečnost: **volání funkce `send()` nastavuje hodnotu naposledy použitého výrazu `yield`**. Můžete si to představit tak, že poslední výraz `yield` předal hodnoty volající funkci, načež zastavil své vykonávání (to už jsme si říkali). Další zavolání dané funkce pokračuje tím, že nejprve vrátí hodnotu, jenomže to je již hodnota dodaná následně volanou funkcí `next()` nebo `send()`, přičemž funkce `send()` předává hodnotu zadanou ve svém argumentu, kdežto funkce `next()` zadává hodnotu `None`.

Výpis 10.51: *Definice generátorové funkce `gye()` demonstrující chování funkce používající hodnotu výrazu `yield`*

```
def gye(up_to):
    """Generátorová funkce pracující s hodnotou výrazu yield."""
    print('Generátor byl odstartován s limitem', up_to)
    acc = ''    # Akumulátor obdržených hodnot
    ret = ''    # Text vrácený volající funkci next() či send()
    get = ''    # Obdržená hodnota
    log = 'Start\n' # Záznam průběhu
    def logf(ch):   # Pomocná logovací funkce
        nonlocal log
        log += f'{ch}:{i=}, {acc=}, {ret=}, {get=}»\n'
    for i in range(up_to):
        logf('A')
        get = yield (ret := '(' + str(i) + ':' + acc + ')')
        logf('\nB')
        if get :
            acc += get[1]   # Zapamatuj si z obdržené hodnoty index
            logf('C')
        logf('D')
    print('Generátor byl ukončen\n')
    logf('\nKonec: ')
    print(log)
```

Pro demonstraci popsaného chování jsem připravil demonstrační program, v němž se snažím ukázat různé možnosti použití. Ve výpisu 10.51 je definice příslušné generátorové funkce a ve výpisu 10.52 je záznam z jejího použití.

Ve výpisu 10.51 je na řádcích 8–10 definována pomocná funkce, která průběžně do proměnné `log` informace o hodnotách lokálních proměnných v jednotlivých etapách zpracování dané funkce. Všimněte si, že proměnná `log` je v ní definována jako `nonlocal`, protože potřebujeme měnit hodnotu lokální proměnné ve vnější funkci. Ostatní proměnné se pouze čtou, takže jsou přirozeně viditelné a nemusíme se s nimi nijak zvlášť zabývat.

Do proměnné `log` zapisujeme hned po spuštění kódu (řádek 7), před použitím výrazu `yield` (řádek 12), po jeho použití (řádek 14), po modifikaci akumulátoru (řádek 17), na konci těla cyklu `for` (řádek 18) a před ukončením běhu dané funkce (řádek 20), přičemž na rozloučenou funkce celý záznam vytiskne.

Možná se budete ptát, proč se logovací záznamy ukládají a proč se hned netisknou. Důvodem je řádek 2 ve výpisu 10.52. Z něj vyplývá, že byla funkce volána při každém průchodu cyklem dvakrát. Poprvé v hlavičce cyklu při přiřazování hodnoty parametru `z`, podruhé pak ve třetím argumentu funkce `print()`, kdy byla volána metoda `send()`.

Kdybychom logovací záznamy rovnou tiskli, tak bychom tiskli uprostřed přípravy parametru pro funkci tisku, a to by mohlo leckoho mást.

Pojďme se nyní podívat na záznam použití vy výpisu 10.52. První věc, na níž bych chtěl upozornit, je, že se kód generátoru nespustil hned po zavolání generátorové funkce, tj. po zadání příkazu na řádku 1, ale až po vlastním spuštění při vykonávání cyklu. Teprve poté se totiž objevuje hlášení o tom, že generátor byl odstartován.

Pak probíhal cyklus z hodnot vypisovaných v pravých závorkách za dvojtečkou si můžete odvodit, že hodnota zadaná jako argument funkce `send()` ovlivní už hodnotu, kterou nám generátor po zavolání této funkce vrací. Jak to? Na to by nám měl odpovědět záznam v proměnné `log`. Pojďme si jej projít.

Záznam začíná na řádku 10 inicializací za níž následuje zápis z řádku 12 (zápis `A`), tj. před vyhodnocováním výrazu `yield`. Za ním následuje prázdný řádek symbolizující, že nyní pracoval kód, jemuž byla před chvílí předána požadovaná hodnota.

Poté byl generátor požádán o další hodnotu, a to (jak víme z řádku 2 výpisu 10.52) zavoláním funkce `send()`, které volající program předal v argumentu aktuální hodnotu parametru cyklu. Kód pokračoval v přerušeném vykonávání příkazu na řádku 13 (řádek s výrazem `yield`) a obdrženou hodnotu vložil do proměnné `get`, jak se můžeme přesvědčit v zápisu `B` na řádku 13. Zápis `C` na řádku 14 pak oznamuje úpravu hodnotu akumulátoru (proměnná `acc`). Zápis `D` na řádku 15 pak pouze oznamuje ukončení těla cyklu a zápis `A` na řádku 16 spuštění dalšího běhu cyklem se zvětšenou hodnotou parametru cyklu.

Řádky 18–20 obsahují o jeden zápis méně, protož tentokrát aktivovala generátor funkce `next()`, která předala hodnotu `None`, takže se tělo příkazu `if` na řádcích 15–17 vůbec neprovedlo.

Výpis 10.52: *Použití generátorové funkce* `gye()`

```
>>> g = gye(6)
>>> for z in g:  print(z, '###', g.send(z))

Generátor byl odstartován s limitem 6
(0:) ### (1:0)
(2:0) ### (3:02)
(4:02) ### (5:024)
Generátor byl ukončen

Start
A:i=0, acc='', ret='', get=''»

B:i=0, acc='', ret='(0:)', get='(0:)'»
C:i=0, acc='0', ret='(0:)', get='(0:)'»
D:i=0, acc='0', ret='(0:)', get='(0:)'»
A:i=1, acc='0', ret='(0:)', get='(0:)'»

B:i=1, acc='0', ret='(1:0)', get=None»
D:i=1, acc='0', ret='(1:0)', get=None»
A:i=2, acc='0', ret='(1:0)', get=None»

B:i=2, acc='0', ret='(2:0)', get='(2:0)'»
C:i=2, acc='02', ret='(2:0)', get='(2:0)'»
D:i=2, acc='02', ret='(2:0)', get='(2:0)'»
A:i=3, acc='02', ret='(2:0)', get='(2:0)'»

B:i=3, acc='02', ret='(3:02)', get=None»
D:i=3, acc='02', ret='(3:02)', get=None»
A:i=4, acc='02', ret='(3:02)', get=None»

B:i=4, acc='02', ret='(4:02)', get='(4:02)'»
C:i=4, acc='024', ret='(4:02)', get='(4:02)'»
D:i=4, acc='024', ret='(4:02)', get='(4:02)'»
A:i=5, acc='024', ret='(4:02)', get='(4:02)'»

B:i=5, acc='024', ret='(5:024)', get=None»
D:i=5, acc='024', ret='(5:024)', get=None»

Konec: :i=5, acc='024', ret='(5:024)', get=None»

>>>
```

Tak bychom mohli projít celý záznam. Snad už je nyní práce s hodnotami poskytovanými výrazem `yield` jasnější.

§