07 map, filter in reduce

January 28, 2024

1 Dva, trije strici iz ozadja

Python ni imel izpeljanih seznamov od vedno. Dobil jih je, ko je imel okrog 9 let, večine študentov pa še ni bilo med nami. Z verzijo 2.0, leta 2000.

Predtem sta tej sceni vladali dve funkciji: `map` in `filter`.

1.1 Map

Funkcija `map` kot argument prejme funkcijo in nekaj, prek česar je možno nagnati zanko. Vsak element tega, nečesa, "premapira" čez funkcijo. Če imamo

[1]:
```
from math import sqrt

k = [9, 25, 16, 81]
```

bo `map(sqrt, k)` vrnil korene vseh števil v `k`:

[2]:
```
for x in map(sqrt, k):
    print(x)
```

Funkcija `map` dela, približno tole:

[3]:
```
def map(func, s):
    t = []
    for x in s:
        t.append(func(x))
    return t
```

Do Pythona 3.0 je funkcija `map` v resnici vračala seznam, od različica 3.0 naprej pa vrne *iterator*. Za tiste, ki ne veste, kaj je to: vede se kot seznam, samo da ni; čezenj lahko gremo z zanko `for`. Za tiste, ki ne veste, pa bi radi izvedeli: preberite zapiske o generatorjih in iteratorjih. Za tiste, ki veste: ja, takšen:

[4]:
```
def map(func, s):
    for x in s:
        yield func(s)
```

Funkcijo `map` od Pythona 2.0 naprej uporabljamo zelo redko. `map(func, s)` je isto kot `(func(x) for x in s)`. Prednost novejše različice je v tem, da

- ne zahteva funkcije, temveč izraz; generatorja `(x ** 2 for x in s)` ne moremo prepisati v `map(**2, s)`, temveč potrebujemo lambdo: `map(lambda x: x ** 2, s)`;
- je `map` počasnejši, ker vedno kliče funkcijo, medtem ko je novejši zapis, generator, ne (vsak, dokler lahko vse opravimo z izrazom).

Osebno `map` rad uporabim, kadar imam funkcijo ravno pri roki in kadar izgleda sintaktično lepše.

Se pravi: redko.

1.2 Filter

Funkcija `filter` je druga funkcija, ki so jo izpeljani seznami spravili ob delo. `filter(func, s)` vrne vse tiste elemente `s`, pri katerih `func` vrne `True`.

[5]:
```
def vsebuje_i(s):
    return "i" in s

imena = ["Ana", "Berta", "Cilka", "Dani", "Ema"]

for x in filter(vsebuje_i, imena):
    print(x)
```

To je seveda isto kot `(x for x in imena if vsebuje_i(x))`, kar je tako ali tako le bolj zapletena različica `(x for x in imena if "i" in x)`. Resnici na ljubo tudi `filter` ne potrebuje poprej definirane funkcije, saj bi lahko pisali `filter(lambda x: "i" in x, imena)`. Vendar je očitno, zakaj filtra ne vidimo več velikokrat.

Izpeljani seznami, slovarji, množice in generatorji v enem zamahu naredijo oboje, mapirajo in filtrirajo.

1.3 Reduce

Funkcija `reduce` je edina iz te družbe, ki ni ostala brezposelna. No, hkrati pa tudi najmanj uporabna od njih, saj Python ni ravno jezik za te hece. Mogoče je tudi to razlog, da jo dobimo v modulu `functools` in ne kar tako, na prostem.

`reduce(func, s)` je nekako ekvivalenten temu `func(func(func(func(s[0], s[1])), s[2]), s[3]), s[4])` - če je `s` seznam s petimi elementi. Ali, v kodi (ki sicer ne zna vsega, kar zna `reduce`):

[6]:
```
def reduce(func, s):
    acc = s[0]
    for x in s[1:]:
        acc = func(acc, x)
    return acc
```

Po domače: `reduce` pokliče funkcijo na prvih dveh elementih, nato na rezultatu tega klica in tretjem elementu, nato na rezultatu tega klica in četrtem elementu… Spremenljivko `acc` pa smo poimenovali po njeni vlogi: akumulator.

Če vemo, kaj so iteratorji in kaj počne `next`, znamo bolj natančno (če ne, pa nič narobe, tudi gornje je dovolj dobro za razumevanje, ki ga potrebujemo za uporabo funkcije):

[7]:
```
def reduce(func, s, acc=None):
    t = iter(s)
    if acc is None:
        acc = next(t)

    for x in t:
        acc = func(acc, x)
    return acc
```

Z `reduce` se da početi zanimive stvari. Pripravimo si nekaj funkcij (ki bi lahko bile tudi lambde, ampak recimo, da jih ne znamo pisati).

[8]:
```
def sestej(a, b):
    return a + b

def zmnozi(a, b):
    return a * b

def vrni_vecjega(a, b):
    if a > b:
        return a
    else:
        return b

def oba_resnicna(a, b):
    return a and b
```

Pripravimo si še priložnostni seznam števil.

[9]:
```
s = [4, 2, 6, 3]
```

Z `reduce` lahko zdaj izračunamo vsoto elementov seznama

[10]:
```
reduce(sestej, s)
```

[10]: 15

produkt

[11]:
```
reduce(zmnozi, s)
```

[11]: 144

in poiščemo največji element

[12]:
```
reduce(vrni_vecjega, s)
```

[12]:
```
6
```

mimogrede pa še 10!, se pravi produkt števil do 10

[13]:
```
reduce(zmnozi, range(1, 11))
```

[13]:
```
3628800
```

Če imamo seznam `True`-jev in `False`-ov, lahko z `reduce` izračunamo njegovo konjunkcijo (`and` prek vseh elementov‘).

[14]:
```
reduce(oba_resnicna, [True, True, True, True, True])
```

[14]:
```
True
```

[15]:
```
reduce(oba_resnicna, [True, True, True, False, True])
```

[15]:
```
False
```

Imenitna reč, problem je le, da se nam teh funkcij ne da definirati vnaprej, Pythonove lambde, s katerimi lahko funkcijo definiramo kar sproti, znotraj klica `reduce`, pa so zelo kilave in tudi nikoli ne bodo drugačne kot kilave.

1.4 Ozadje stricev iz ozadja

Funkcije `map`, `filter` in `reduce` - pri čemer se slednja v drugih jezikih bolj pogosto kot ne imenuje `fold` - so osnovni elementi funkcijskega programiranja.