02b zanke

January 28, 2024

0.1 Prva zanka

Kaj znamo narediti doslej: znamo napisati program, ki kaj vpraša, kaj računa in kaj izpiše. Naši programi se izvajajo od začetka do konca, s tem da lahko vmes preskočijo kak blok ukazov, ki se nahajajo znotraj kakega `if`, `else` ali `elif`. Večino časa pa smo preživeli ob tem, kako napisati pogoj, se pravi, izraz, katerega vrednost je lahko resnično ali neresnično, `True` ali `False`. Zdaj pa se bomo naučili napisati programe, v katerih se del programa ponavlja.

V večini programskih jezikov to naredimo z nečim, čemur mi rečemo zanka, Avstralci (razen aboriginov) pa *loop*. Jeziki imajo navadno več vrst zank, pogosto tri. Python premore samo dve in danes bomo spoznali prvo od njiju: zanko **while**.

Tisočletna tradicija veleva, da mora biti začetnikov prvi program z zanko program, ki šteje do deset. V Pythonu je videti takole

[ ]:
```
x = 1
while x <= 10:
    print(x)
    x = x + 1
print('Pa sva preštela')
```

Zanka `while` je po svoje podobna stavku `if`. Medtem ko se stavki znotraj `if` izvedejo, *če* je pogoj resničen, se stavki znotraj `while` ponavljajo, *dokler* je pogoj resničen. "If" je "če" in "while" je "dokler", to vedo celo učiteljice angleščine. (Mimogrede, če že štejemo, `while` je zadnja, enajsta rezervirana beseda danes. Smo že na tretjini!) Gornji program lahko torej kar dobesedno prevedemo iz Pythona v slovenščino:

[ ]:
```
postavi x na 1
dokler je x manjši ali enak 10:
    izpiši vrednost x
    povečaj x za 1
izpiši 'Pa sva preštela'
```

K temu ni potrebna več nobena razlaga, ne?

Ker je ravno pravi trenutek, povejmo za priročno bližnjico. Tale `x = x + 1` smo "prevedli" v "povečaj x za 1". V resnici nismo napisali tega, računalniku smo naročili, naj izračuna, koliko je `x + 1` in to shrani nazaj v `x`. Ker pa to reč tolikokrat potrebujemo in ker je lepo, da se stvar napiše tako, kot se misli - misli pa se "povečaj x za 1" -, obstaja tudi praktičnejši zapis: `x += 1`. Točneje, napisali smo "k x prištej 1". Rečemo lahko tudi, `x += 6` ali `x += 1.13`; kaj to pomeni, je menda jasno. Podobno moremo pisati tudi `x -= 1` (zmanjšaj `x` za 1) ali, recimo `x *= 2`, podvoji `x` in `x`

`/= 10`, zdesetkaj `x`. Pazite: `+=` (in vse druge kombinacije) se štejejo kot ena beseda in med znaka `+` in `=` ne smemo napisati presledka.

Tisti, ki veste še kaj več: `x++` boste v Pythonu zaman iskali. Nima ga, iz več razlogov, ki pa so malo pregloboki za nas tule.

0.2 Kaj je domneval Collatz

Vzemimo poljubno število in z njim počnimo tole: če je sodo, ga delimo z 2, če je liho, pa ga pomnožimo s 3 in prištejmo 1. To ponavljamo, dokler ne dobimo 1.

Za primer vzemimo 12. Ker je sodo, ga delimo z 2 in dobimo 6. 6 je sodo, torej ga delimo z 2 in dobimo 3. 3 je liho, torej ga množimo s 3 in prištejemo 1 - rezultat je 10. 10 je sodo, zato ga delimo z 2 in dobimo 5… Celotno zaporedje je 12, 6, 3, 10, 5, 16, 8, 4, 2, 1. Ko pridemo do 1, se ustavimo.

Matematiki, ki vedno radi počnejo koristne reči, si belijo glavo z vprašanjem, ali se reč vedno izteče ali pa se lahko kdaj tudi zacikla tako, da se zaporedje ponavlja in ponavlja v nedogled. Lothar Collatz, konkretno je že od leta 1937 domneval, da se zaporedje vedno izteče. Kot pravi njegov življenjepis, se je njegovo domnevanje končalo leta 1990, matematiki pa domnevajo domnevo še naprej. Eden sicer najbolj neustrašnih matematikov 20. stoletja, couch surfer Erdos, je na to temo rekel, da matematika za takšna vprašanja še ni zrela.

Naši cilji bodo skromnejši: napisali bomo program, ki mu bo uporabnik vpisal število in program bo izpisal zaporedje, kot je bilo gornje.

Najprej povejmo kar po slovensko:

[ ]:
```
stevilo = kar vpiše uporabnik
dokler stevilo != 1:
    če je število sodo:
        deli število z 2
    sicer:
        pomnoži število s 3 in prištej 1
```

Tole skoraj že znamo prevesti v Python (prevod bo pravzaprav dobeseden), odkriti moramo le, kako preveriti, ali je število sodo. Če kdo misli, da tudi za to obstaja funkcija, se, izjemoma, moti. Pač pa znamo izračunati ostanek po deljenju, konkretno, ostanek po deljenju z 2. Če je enak 0, je število sodo.

Prevedimo torej one slovenske besede v pythonovske.

[ ]:
```
n = int(input("Vnesi število: "))
while n != 1:
    print(n)
    if n % 2 == 0:
        n //= 2
    else:
        n = n * 3 + 1
print(1)
```

Ne spreglejte, da smo uporabili celoštevilsko deljenje. Če bi uporabili navadnega, `n /= 2`, bi se `n` spremenil v necelo število (`float`) in v izpisu bi se pojavile zoprne decimalke (za vsakim številom

bi pisalo še .0).

Popaziti je potrebno, da izpišemo tako prvo kot zadnje število, zato bomo potrebovali še en `print` izven zanke: po zanki izpišemo še zadnji člen zaporedja, 1. Namesto `print(1)` bi lahko napisali tudi `print(n)` - tako ali tako vemo, da je `n` enak 1, saj program sicer ne bi prišel do tam, do koder je prišel, namreč onstran zanke, ki teče, dokler je `n` različen od 1.

Dodajmo še nekaj: na koncu naj se izpiše, koliko členov ima zaporedje.

[ ]:
```
n = int(input("Vnesi število: "))
clenov = 1
while n != 1:
    clenov += 1
    print(n)
    if n % 2 == 0:
        n //= 2
    else:
        n = n * 3 + 1
print(1)
print("Zaporedje, ki se začne z", n, "ima", clenov, "členov")
```

Ups. V zanki smo pokvarili `n`, spremenil smo ga v 1. To bomo rešili tako, da shranimo začetno število v še eno spremenljivko.

[ ]:
```
zac = n = int(input("Vnesi število: "))
clenov = 1
while n != 1:
    clenov += 1
    print(n)
    if n % 2 == 0:
        n //= 2
    else:
        n = n * 3 + 1
print(1)
print("Zaporedje, ki se začne z", zac, "ima", clenov, "členov")
```

Število smo si zapomnili v prvi vrstici, shranili smo ga v `zac`. (V tem programu tudi prvič vidimo, da lahko prirejanja kar nizamo - pisati smemo `a = b = c = 42`.) Ustrezno popravimo še izpis v zadnji vrsti.