02a pogoji

January 28, 2024

0.1 Pogojni stavki

Začnimo s preprostim programom za izračun indeksa telesne teže.

[1]:
```
teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
```

```
Teža: 60
Telesna višina: 170
Indeks vaše telesne teže je 0.0020761245674740486
```

Želimo si, da bi program povedal še, ali je uporabnik slučajno preobilen. Prvi, seveda napačni poskus, bi bil takšen.

[ ]:
```
teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
print("Potrebno bo shujšati!")
```

Program v tej obliki bi, v širši rabi, povzročal anoreksijo, saj vse po vrsti, vključno z najstnicami, pošilja na hujšanje. Zadnjo vrstico, poziv k dieti, mora izvesti le, če je `bmi` večji od 25 (ta meja namreč velja za mejo debelosti, oziroma, politično korektno, prekomerne telesne teže).

To storimo tako, da pred vrstico dodamo pogoj. Ta se glasi "če bmi $>$ 25", le da ga moramo napisati v angleščini.

[ ]:
```
teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
```

Ne spreglejmo dveh pomembnih reči. Prva: na konec pogoja smo postavili dvopičje. To ne služi le za okras: dvopičja ne smemo izpustiti. Python na tem mestu zahteva dvopičje.

Druga: prejšnjič sem prepovedal pisanje presledkov na začetku vrstice … razen takrat, ko vam bom zapovedal, da jih pišite. No, napočil je ta trenutek: zapovedujem vam. Poglejte, kako sem oblikoval

program: vrstico za `if`-om sem nekoliko zamaknil. Koliko, ni pomembno, pomembno je samo, da sem vse zamaknil enako. (Neobvezno pravilo: zamik naj bo velik štiri presledke. Ni obvezno, a če boste uporabljali štiri presledke, boste pisali enako kodo kot drugi. In nikoli nikoli ne uporabljajte tabulatorjev, ti naredijo zmedo!)

Zamaknjenih vrstic bi lahko bilo še več. Vse se izvedejo le, če je pogoj izpolnjen.

[ ]:
```
teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
    print("Da, gospod ali gospa, preveč vas je skupaj.")
print("Pregled je končan, oglasite se spet čez dva tedna.")
```

Tako kot smo zastavili zdaj, se `print`a, ki sta zamaknjena, izvedeta le, če je oseba preobilna. Zadnji `print` ni več znotraj pogoja, zato se izvede v vsakem primeru, za suhe in debele.

Tale razlaga je skoraj nepotrebna, saj je to, kdaj se izvede kaj, očitno že kar iz oblike programa. (To je razlog, zakaj je Python tako simpatičen jezik za začetnike: programe v njem je veliko lažje brati kot programe v kakem bolj kriptičnem jeziku.)

Program bi bil še bolj simpatičen, če bi tiste, ki niso predebeli, pohvalil.

[ ]:
```
teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
    print("Da, gospod ali gospa, preveč vas je skupaj.")
else:
    print("Odlično, le še naprej jejte toliko kot doslej!")
print("Pregled je končan, oglasite se spet čez dva tedna.")
```

Pogojnemu stavku (ups, nov izraz; računalnikarji nekaterim rečem rečemo stavek, ker pa vas tule ne bom mučil s terminologijo, si pod besedo *stavek* predstavljajte preprosto "kos programa") lahko sledi še `else`, ki vsebuje tisto, kar naj se zgodi, če pogoj ni izpolnjen.

Znotraj pogojnih stavkov seveda lahko pišemo tudi druge reči, ne le `print`ov. Takole: uporabniki programa se bodo pogosto zmotili in namesto višine v metrih vpisali višino v centimetrih, zato jih bo program obtožil težke anoreksije. To lahko popravimo preprosto tako, da program še pred računanjem BMI preveri, ali je višina slučajno večja od, recimo, treh metrov. V tem primeru sklepa, da je uporabnik podal višino v centimetrih, zato številko, ki jo je vpisal uporabnik, deli s 100.

Začetek programa tako postane takšen.

[ ]:
```
teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
if visina > 3:  # Predpostavimo, da ni Hagrid
```

```
    visina = visina / 100
bmi = teza / visina ** 2
```

Naprej pa gre tako kot prej.

Na prvem predavanju smo omenjali *rezervirane besede*, to je, besede, ki ne morejo biti imena spremenljivk. Med drugim sem pokazal besedo `if` - napisal sem `if = 1` in Python se je pritožil, da mu ni niti približno jasno, kaj hočem povedati.

Zdaj vidimo, zakaj. Besedi `if` mora slediti nek pogoj. In `= 1` ni pogoj.

Nekoliko off topic: tako zamaknjenim delom programa pravimo, kot ste me že večkrat slišali reči, *bloki*. Bloke poznajo vsi "normalni" programski jeziki, le označujejo jih različno. Pogost način označevanja je označevanje z zavitimi oklepaji: v Cju (in iz njega izpeljanih oz. njemu podobnih jezikih C++, Java, C#, JavaScript, Php…) bi bili naši `if`i videti takole (poleg oklepajev je razlik še nekaj, vendar nas ne zanimajo in jih spreglejte):

[ ]:
```
if (bmi > 25) {
    printf("Potrebno bo shujšati");
    printf("Da, gospod ali gospa, preveč vas je skupaj.");
}
else {
    printf("Odlično, le še naprej jejte toliko kot doslej!");
}
printf("Pregled je končan, oglasite se spet čez dva tedna.");
```

Kaj je lepše, je stvar okusa in predmet številnih verskih vojn.

Za drugoverce, se pravi heretike, ki so vam zaviti oklepaji bolj všeč od zamikanja: najpogostejša napaka v naslednjih dveh tednih bo, da boste pozabljali dvopičje. Če se vaš program noče izvesti zaradi sintaktične napake, preverite dvopičja. Po dveh tednih se boste navadili.

Če smo že ravno pri verskih vprašanjih: v Pythonu ne pišemo oklepajev, kjer niso potrebni … razen tam, kjer povečajo čitljivost. Noben greh ni napisati `teza / (visina ** 2)`, čeprav bi se vse izračunalo popolnoma enako tudi, če oklepajev ne bi bilo. Potenciranje ima sicer prednost pred deljenjem, vendar je to, če napišemo še oklepaje, jasneje. V pogojnih izrazih pa oklepajev ne pišemo. Napisali bomo torej `if bmi > 25` in ne `if (bmi > 25)`, kot zahtevajo nekateri drugi jeziki. Oklepajev vam nihče ne brani, vendar vas bodo v teh, Pythonovskih krajih, čudno gledali, če boste preveč strašili z njimi. Ko programiraš v nekem jeziku, programiraj, kot se programira v tem jeziku. Pisati oklepaje v pogojih v Pythonu, je, kot da bi po angleško govorili s fransoskim aksentom samo sato, ker ze vam zdi fransoščina lepša od anglaise.

Še ena estetska zadeva: za dvopičjem vedno pojdite v novo vrsto. Kot boste zvedavi kmalu odkrili, Python sicer dovoli, da napišete tudi

[ ]:
```
if bmi > 25: print ("Potrebno bo shujšati")
```

vendar to vodi v nepregledne programe. To se ne dela.

Tule pa je še nekaj primerov napačno zamikanih programov.

[ ]:
```
if bmi > 25:
    print("Potrebno bo shujšati!")
  print("Da, gospod ali gospa, preveč vas je skupaj.")
else:
    print("Odlično, le še naprej jejte toliko kot doslej!")
print("Pregled je končan, oglasite se spet čez dva tedna.")
```

Tretja vrstica mora imeti enak zamik kot druga, sicer Python ne more vedeti, ali je še znotraj pogoja ali ne.

[ ]:
```
if bmi > 25:
    print("Potrebno bo shujšati!")
      print("Da, gospod ali gospa, preveč vas je skupaj.")
else:
    print("Odlično, le še naprej jejte toliko kot doslej!")
print("Pregled je končan, oglasite se spet čez dva tedna.")
```

Ne, tudi v to smer ne smemo. Tu se Python vpraša, zakaj je tretja vrstica zamaknjena še bolj kot druga - smo pozabili še kakšen pogoj ali kaj?

[ ]:
```
if bmi > 25:
print("Potrebno bo shujšati!")
print("Da, gospod ali gospa, preveč vas je skupaj.")
```

Blok za stavkom `if` mora biti zamaknjen.

[ ]:
```
if bmi > 25:
    print("Potrebno bo shujšati!")
    print("Da, gospod ali gospa, preveč vas je skupaj.")
  else:
    print("Odlično, le še naprej jejte toliko kot doslej!")
print("Pregled je končan, oglasite se spet čez dva tedna.")
```

`if` in `else` morata biti poravnana. Kmalu bomo videli programe z več `if`i in `else`i; zamiki bodo potrebni, da bomo vedeli, kateri `else` sodi h kateremu `if`.

Če poskušamo zagnati kateregakoli od naštetih programov, bo Python javil sintaktično napako. Za razliko od običajnih napak, ki jih boste delali, in ko bodo programi naredili vsaj nekaj malega in potem crknili, ta ne bo niti trznil, temveč kar takoj javil napako. Spodnji program pa je sintaktično pravilen. Python ga lahko izvede. Najbrž pa to ni to, kar smo hoteli, saj se zadnji `print` izvede le, če pogoj ni izpolnjen.

[ ]:
```
if bmi > 25:
    print("Potrebno bo shujšati!")
    print("Da, gospod ali gospa, preveč vas je skupaj.")
else:
    print("Odlično, le še naprej jejte toliko kot doslej!")
    print("Pregled je končan, oglasite se spet čez dva tedna.")
```

0.2 Gnezdeni pogoji

Do sem vse lepo in prav, vendar: kaj bomo z anoreksičnimi? Program bi jim mirno svetoval, naj še naprej jedo toliko kot doslej (torej: premalo). Ne: pri takšnih, ki imajo BMI manjši od 18.5, moramo zastokati, naj vendar že začnejo jesti.

[ ]:
```
teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
    print("Da, gospod ali gospa, preveč vas je skupaj.")
else:
    if bmi < 18.5:
        print("Trlica!")
        print("Zunaj so avtomati s sendviči in čokolado. Med pavzo si le␣
 ↪postrezite!")
    else:
        print("Odlično, le še naprej jejte toliko kot doslej!")
print("Pregled je končan, oglasite se spet čez dva tedna.")
```

Vidimo? Znotraj `if`a ali `else`a smemo ugnezditi nov pogoj. Zamikamo ga lepo naprej.

Da bodo reči še nekoliko daljše (a poučnejše), dodajmo še en `print`.

[ ]:
```
teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
    print("Da, gospod ali gospa, preveč vas je skupaj.")
else:
    print("Predebeli ravno niste.")
    if bmi < 18.5:
        print("Trlica!")
        print("Zunaj so avtomati s sendviči in čokolado. Med pavzo si le␣
 ↪postrezite!")
    else:
        print("Odlično, le še naprej jejte toliko kot doslej!")
print("Pregled je končan, oglasite se spet čez dva tedna.")
```

Zdaj program vsem, ki niso predebeli, napiše dobro novico, da niso predebeli. Šele nato se loti preverjanja, da niso slučajno presuhi.

Tule mimogrede (prvič - a še velikokrat bomo) posvarimo pred ponavljanjem kode. Gornji program bi lahko napisali tudi tako:

```
[ ]: teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
    print("Da, gospod ali gospa, preveč vas je skupaj.")
else:
    if bmi < 18.5:
        print("Predebeli ravno niste.")
        print("Trlica!")
        print("Zunaj so avtomati s sendviči in čokolado. Med pavzo si le␣
 ↪postrezite!")
    else:
        print("Predebeli ravno niste.")
        print("Odlično, le še naprej jejte toliko kot doslej!")
print("Pregled je končan, oglasite se spet čez dva tedna.")
```

Ali celo tako.

```
[ ]: teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
    print("Da, gospod ali gospa, preveč vas je skupaj.")
    print("Pregled je končan, oglasite se spet čez dva tedna.")
else:
    if bmi < 18.5:
        print("Predebeli ravno niste.")
        print("Trlica!")
        print("Zunaj so avtomati s sendviči in čokolado. Med pavzo si le␣
 ↪postrezite!")
        print("Pregled je končan, oglasite se spet čez dva tedna.")
    else:
        print("Predebeli ravno niste.")
        print("Odlično, le še naprej jejte toliko kot doslej!")
        print("Pregled je končan, oglasite se spet čez dva tedna.")
```

To ni dobra ideja, ker je program daljši, manj pregleden, težje mu je slediti, težje ga je spreminjati... Vedno se izogibajte ponavljanju.

0.3 Sicerče

Da se ne izgubimo, spet malo skrajšajmo program.

```
[ ]: teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
else:
    if bmi < 18.5:
        print("Trlica!")
    else:
        print("Odlično, le še naprej jejte toliko kot doslej!")
```

Pogojni stavki se nam lahko hitro zagnezdijo precej globoko in vrstice zbežijo nekam daleč proti desni. (Da ne govorimo o tem, da pogojni stavki niso niti približno edina reč, za katero bomo morali zamikati vrstice.) Ker so situacija, kakršna je zgornja, kar pogoste, imajo nekateri programski jeziki - in Python je med njimi - poleg `if` in `else` še `elseif` oziroma `elif`. (Še ena rezervirana beseda!) V Pythonu se uporablja slednja različica, `elif`. V gornjem programu ga uporabimo takole:

```
[ ]: teza = float(input("Teža: "))
visina = float(input("Telesna višina: "))
bmi = teza / visina ** 2
print("Indeks vaše telesne teže je", bmi)
if bmi > 25:
    print("Potrebno bo shujšati!")
elif bmi < 18.5:
    print("Trlica!")
else:
    print("Odlično, le še naprej jejte toliko kot doslej!")
```

Program se zdaj prebere še lepše kot prej: če je BMI prevelik, pozovemo k hujšanju, sicerče je premajhen, ozmerjamo dotično najstnico s trlico, sicer pa izrečemo pohvalo.

Po izkušnjah je `elif` za študente huda, nenavadna skušnjava: nekateri ga pogosto uporabljajo namesto `else`a. Recimo, da bi hoteli sprogramirati skrajšano pesimistično različico, po kateri so vsi, ki niso debeluhi, trlice. Radi bi torej tole:

```
[ ]: if bmi > 25:
    print("Potrebno bo shujšati!")
else:
    print("Trlica!")
```

Mnogi bi (iz meni neznanega razloga) naredili takole:

```
[ ]: if bmi > 25:
    print("Potrebno bo shujšati!")
elif bmi < 25:
    print("Trlica!")
```

Python ni pozabil `if`a in bo tudi, če napišemo le `else`, čisto dobro vedel, kakšen je pogoj zanj.

Drugi pogoj je nepotreben; če `bmi` ni večji od 25, je pač očitno manjši. (No ja, lahko bi bil tudi enak. A sporočilo smo, upam, razumeli.)

Torej: ne zamenjujte `else` brez potrebe z `elif`.

0.4 Pogoji

Zdaj pa povejmo nekaj splošnejšega o pogojih. Kakšni vse so lahko? Najprej, pogoj je izraz. Torej nekaj, kar se da izračunati (po naši garažni definiciji izraza). Doslej so bili vsi naši izrazi nekako številski, aritmetični: v njih so nastopala števila in le-ta smo seštevali, množili, odštevali in kar se še takega dela s števili. Pogoji pa so logični izrazi. (V resnici se "logičnost" in "številskost" lahko poljubno prepletata, a to bomo mimogrede odkrili sproti.)

Rezultat logičnega izraza je logična vrednost. Logične vrednosti niso števila (torej niso `int` ali `float`) in niso nizi, torej so nekaj četrtega. Podatkovni tip, ki hrani logične vrednosti, se imenuje `bool` (iz Boolean, po slovensko Booleov). Medtem ko je različnih števil (`int`, `float`) kar veliko (koliko, vam bodo povedali kolegi matematiki, ko bodo v predavalnico privlekli Cantorja in njegov trop alefov), ima tip `bool` lahko le eno od dveh vrednosti, `True` ali `False`. Tudi `True` in `False` rezervirani besedi. (Če prav štejemo, smo naleteli že na sedem rezerviranih besed od 35 - `if`, `else`, `elif`, `True` in `False`, na vajah oziroma ob domači nalogi pa še `from` in `import`.)

S števili že znamo računati. Z nizi tudi malo. Kako pa računamo z logičnimi vrednostmi? Menda jih ne moremo kar tako seštevati in odštevati?

Za logične izraze imamo (predvsem) tri operatorje: `or`, `and` in `not`. Kaj pomenijo, je menda jasno, če je kdo v dvomih, pa naj se pouči ob prvem predavanju iz Diskretnih struktur.

[ ]:
```
>>> True and True
True
>>> True and False
False
>>> False or True
True
>>> not False
True
>>> False or not False
True
```

(`and`, `or` in `not` so rezervirane besede. 10 od 35!)

Operator `not` ima prednost pred `and`, `and` pa pred `or`. Spet si lahko pomagamo z oklepaji: `a or b and c` pomeni isto kot `a or (b and c)` in bo resničen (`True`), če je resničen `a` ali pa sta resnična `b` in `c`. Izraz `(a or b) and c` pa bo resničen, če sta resnična `a` ali `b` (ali oba), poleg tega pa nujno še `c`. Pri tem seveda predpostavljamo, da `a`, `b` in `c` vsebujejo logične vrednosti.

Da bo reč lažje razumljiva, jo potrenirajmo na prav tem izrazu, vendar tako, da namesto `a`, `b` in `c` oziroma namesto "golih" `True` in `False` pišemo bolj "konkretne" izraze.

[ ]:
```
>>> 352 > 0
True
>>> 5 < 3
```

```
False
>>> 352 > 0 or 5 < 3
True
>>> 352 > 0 or 5 > 3
True
>>> 352 > 0 and 5 > 3
True
>>> 352 > 0 and 5 < 3
False
>>> 352 > 0 and not 5 < 3
True
>>> 10 > 8 or 352 > 0 and 5 < 3
True
>>> 10 > 8 or (352 > 0 and 5 < 3)
True
>>> (10 > 8 or 352 > 0) and 5 < 3
False
>>> (10 > 8 or 352 > 0) and not 5 < 3
True
```

Operatorja < in > pričakujeta na levi in desni kake reči, ki jih je mogoče primerjati; zadovoljna nista le s števili, temveč tudi z, recimo, nizi (primerjala jih bosta po abecedi), ne moreta pa primerjati števila z nizom. Njun rezultat je logična vrednost. Za "večje ali enako" in "manjše ali enako" uporabimo `>=` in `<=`.

Če želimo preveriti enakost dveh števil (ali, v splošnem, enakost dveh stvari), uporabimo dvojni enačaj, `==`. Enojni enačaj je namreč dobil svojo zadolžitev prejšnjo uro, namenjen je prirejanju. Ali sta dve stvari različni, preverimo z `!=`.

[ ]:
```
>>> 1 + 1 == 2
True
>>> 1 + 1 != 2
False
```

Ne zamenjujte dvojnih in enojnih enačajev. Če pomotoma napišemo (a to se nam bo redko zgodilo), `a == 1 + 1` s tem nismo priredili `a`-ju dvojke, temveč smo le preverili, ali je enak 2. Pogostejša napaka pa bo

[ ]:
```
if a = 1:
    print("Vrednost a je ena")
```

Ta program pa je sintaktično nepravilen, saj Python za `if` pričakuje pogoj, mi pa smo napisali prireditveni stavek.

(Omenimo druge jezike: večina tega je enaka v vseh jezikih. Za različnost imajo namesto `!=` nekateri `<>`. Namesto `and`, `or` in `not` imajo mnogi jeziki `&&`, `||` in `!`. V Pythonu so se odločili za "besedne" variante, da so programi zračnejši.)

Zdaj pa Pythonova posebnost: operatorje smemo nizati. Izraz `10 < t < 20` je resničen, če je `t`

med 10 in 20. Izraz `10 < t1 < t2 < 20` je resničen, če sta `t1` in `t2` med 10 in 20, pri čemer je `t1` manjši od `t2`. Izraz `t1 < 10 < t2` je resničen, če je `t1` manjši od 10, `t2` pa večji od 10. `10 < t1 == t2 < 20` je resničen, če sta `t1` in `t2` enaka ter sta nekje med 10 in 20.

Če se torej končno vrnemo k suhcem in debeluhom: če bi se hoteli skoncentrirati le na te, ki so ravno pravšnji, bi lahko napisali, recimo

[ ]:
```
if bmi > 18.5 and bmi < 25:
    print("Čestitamo, ravno pravšnji ste.")
```

Če bi kdo napisal

[ ]:
```
if (bmi > 18.5) and (bmi < 25):
    print("Čestitamo, ravno pravšnji ste.")
```

ali celo

[ ]:
```
if ((bmi > 18.5) and (bmi < 25)):
    print("Čestitamo, ravno pravšnji ste.")
```

Ga bomo čudno gledali (((kdor hoče oklepaje, naj gre raje programirat v Lispu))). Prav po Pythonovsko pa se reče:

[ ]:
```
if 18.5 < bmi < 25:
    print("Čestitamo, ravno pravšnji ste.")
```

0.5 Kurzschluss

Tale tema je nekoliko bolj zapletena, vendar je prav, da tudi pri Programiranju 1 slišite kaj zapletenega. (Pa ne preveč.) No, v resnici pa vam moram tole povedati tudi zavoljo temeljitosti. Da mi ne bo kdo očital, da sem pri poučevanju šlampast. Sploh pa se je na prejšnjih predavanjih pokazalo, da ste pametnejši, kot smo pričakovali, in prav je, da ste za to vsaj malo tepeni.

Wikipedia definira kratek stik v električni napeljavi oz. vezju kot nekaj, kar dovoli elektriki, da potuje po krajši poti, kot je bila mišljena. Po domače, kurzschluss je, ko se v televiziji nekaj zabliska, potem pa namesto, da bi delala, samo še smrdi. (Ravno v času pisanja tega besedila sem imel namreč v dnevni sobi na mizi primerek televizije, ki je popolnoma ustrezal gornjemu opisu, vključno z bliskanjem in smradom.) Namesto da bi elektrika tekla čez transformatorje in druga navitja, steče po bližnjici.

Short-circuit behaviour, lahko bi ga prevedli kot kratkostično vedenje, v programskih jezikih pomeni, da po bližnjici steče izvajanje programa. Pri tem navadno ni ne bliskanja ne smradu, ker je takšno vedenje zaželeno. Za primer poglejmo, kako bi se računala vrednost `(10 < 8 or 352 < 0) and 18574 + 50485 == 69059`. 10 ni manj od 8 in 352 ni manj od 0. `False or False` je `False`. Prvi del izraza, vse, kar je pred `and`-om je `False`. Preden začne računalnik seštevati tiste velike številke desno od `and`-a pa pomisli: komu na čast? Po tem, kar sem naračunal doslej, vem, da imam `False and 18574 + 50485 == 69059`. To bo neresnično v vsakem primeru, ne glede na ono vsoto. Zato drugega dela sploh ne računa, temveč že kar takoj vrne `False`.

Pravilo je torej takšno: če je izraz, ki je levo od `and`, neresničen, vrednosti izraza, ki je desno od `and`, sploh ne računa.

Podobno, le ravno obratno, bližnjico lahko uberemo pri `or`. Kako bi računali vrednost `10 > 8 or 352 > 0`? Najprej ugotovimo, da je 10 res več kot 8. Moramo zdaj res preverjati še, ali je 352 več kot 0? Ne, saj že vemo, da bo izraz resničen. Kako pa je z `10 < 8 or 352 > 0`. Bomo, ko ugotovimo, da ni res, da je 10 manjše od 8, nehali računati? Ne, izraz je še vedno lahko resničen, reši nas lahko drugi del (in tudi v resnici nas). Pač pa bi pri izrazu `10 < 8 and 352 > 0` vrgli puško v koruzo že po prvem neuspehu, saj nas morebitni drugi uspeh ne more rešiti.

Pravilo kratkega stika si lahko zapomnimo na dva načina. Prvi, formalni, je tale: če je izraz levo od `or` resničen, potem onega desno od `or` ne preverjamo, saj že vemo, da bo rezultat resničen. Prav tako, če je izraz levo od `and` neresničen, potem onega desno od njega ne preverjamo, saj že vemo, da bo rezultat neresničen. Drugi način je neformalen: izraz se računa od leve proti desni in samo, dokler je treba.

Študent pa se najbrž že sprašuje: me to res briga? Računalnik naj računa kar in kolikor hoče, meni je pomembno samo, da bo na koncu naračunal prav. V resnici je tako: če kratkega stika ne poznamo, nam ne bo nič hudega. Če ga, pa nam lahko pride zelo zelo prav. Primer pride kmalu.