

Petlje i logički izrazi

Slajdovi za predmet Osnove programiranja

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2014.

Ciljevi

- koncepti konačne i beskonačne petlje pomoću `for` i `while` naredbi
- interaktivna petlja i sentinel petlja korišćenjem `while` naredbe
- end-of-file petlja
- ugnježdene petlje
- Bulova algebra i Bulovi izrazi

Ciljevi

- Bulovi izrazi i `bool` tip podataka
- kreiranje algoritama koji uključuju elemente kontrole toka
- uključujući nizove grananja i ugnježdano grananje

for petlja: podsećanje

- for petlja omogućava **iteraciju** kroz niz vrednosti
for <var> in <sequence>:
 <body>
- indeksna promenljiva var uzima po jednu vrednost iz niza u svakom prolazu petlje
- u svakom prolazu telo petlje izvrši se jednom, za svaku vrednost var

Računanje proseka

- pišemo program koji računa prosečnu vrednost niza brojeva
- trebalo bi da radi sa nizom brojeva bilo koje dužine
- ne moramo da pamtimo sve brojeve, samo da pamtimo tekuću ukupnu sumu i broj brojeva

Računanje proseka 2

- nešto slično smo već sretali
- niz brojeva se može obraditi petljom
- ako ima n brojeva, petlja treba da ima n ciklusa
- treba nam tekući zbir svih dosadašnjih brojeva – koristićemo akumulator

Algoritam za računanje proseka niza brojeva

- 1 unesi broj brojeva n
- 2 inicijalizuj sum na 0
- 3 izvrši n puta:
 - unesi broj x
 - dodaj x na sum
- 4 ispiši prosek kao sum/n

Program za računanje proseka

```
# average1.py

def main():
    n = eval(raw_input("Koliko ima brojeva? "))
    sum = 0.0
    for i in range(n):
        x = eval(raw_input("Unesi broj >> "))
        sum = sum + x
    print "\nProsek je", sum / n
```

- sum je inicijalizovan na 0.0 umesto na 0
- zato će sum/n biti float a ne int!

Program za računanje proseka 2

Koliko ima brojeva? 5

Unesi broj >> 32

Unesi broj >> 45

Unesi broj >> 34

Unesi broj >> 76

Unesi broj >> 45

Prosek je 46.4

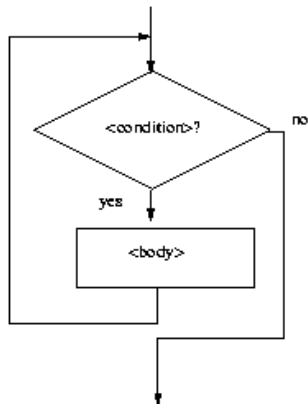
Uslovna petlja

- prethodni program mora unapred da zna koliko ima brojeva
- želimo da program sam vodi računa o tome koliko ima brojeva
- `for` petlja je konačna petlja – ima unapred poznat broj ciklusa

Uslovna petlja 2

- ne možemo da koristimo konačnu petlju ako ne znamo unapred broj ciklusa
- ne znamo koliko ima ciklusa dok se ne unesu svi brojevi
- treba nam **uslovna** petlja – ponavlja telo dok se ne ispuni neki uslov

Uslovna petlja ₃



- uslov se ispituje **na vrhu** petlje
- „petlja sa izlaskom na vrhu“
- telo se može izvršiti 0 puta, 1 put, ili više puta

Primeri while petlje

- primer while petlje koja broji od 0 do 10:

```
i = 0
while i <= 10:
    print(i)
    i = i + 1
```

- isto radi kao i sledeća for petlja:

```
for i in range(11):
    print(i)
```

while vs for

- kod `while` petlje moramo sami da održavamo indeks petlje (`i`)
- **inicijalizujemo** ga na 0 i „ručno“ ga **inkrementiramo** na kraju tela petlje
- kod `for` petlje ovo se odvija automatski

while petlja: oprez

- while petlja je moćnija
- predstavlja izvor mogućih grešaka
- kako radi ovaj kod?

```
i = 0
while i <= 10:
    print(i)
```

while petlja: oprez

- while petlja je moćnija
- predstavlja izvor mogućih grešaka
- kako radi ovaj kod?

```
i = 0
while i <= 10:
    print(i)
```

- ovo je primer **beskonačne petlje**

Beskonačna petlja

- šta da radimo ako program uđe u beskonačnu petlju?
- pritisnemo Ctrl-C
- ako ne pomogne, pritisnemo Ctrl-Alt-Del
- ako ne pomogne, resetujemo računar :)

Interaktivna petlja

- beskonačnu petlju možemo koristiti za pisanje **interaktivnih** petlji
- interaktivna petlja omogućava korisniku da ponavlja određeni deo programa neposredno na zahtev
- treba nam i način da evidentiramo koliko brojeva je uneto
- korišćićemo još jedan akumulator – `count`

Interaktivna petlja 2

- u svakom ciklusu petlje, pitaj korisnika da li ima još brojeva za unos
- taj odgovor pre početka mora biti „da“ da bismo ušli u prvi ciklus petlje

```
postavi moredata na "yes"
```

```
while moredata == "yes"
```

```
    učitaj sledeći podatak
```

```
    obradi podatak
```

```
    pitaj korisnika da li ima još podataka
```

Interaktivna petlja ₃

- kombinujemo interaktivnu petlju i akumulatore za sum i count

```
inicijalizuj sum na 0.0
inicijalizuj count na 0
postavi moredata na "yes"
while moredata == "yes"
    unesi broj x
    dodaj x na sum
    dodaj 1 na count
    pitaj korisnika da li ima još podataka
ispiši sum/count
```

Interaktivna petlja 4

```
# average2.py

def main():
    moredata = "da"
    sum = 0.0
    count = 0
    while moredata == 'da':
        x = eval(raw_input("Unesite broj >> "))
        sum = sum + x
        count = count + 1
        moredata = raw_input(
            "Ima još brojeva (da ili ne)? ")
    print "\nProsek je", sum / count
```

Interaktivna petlja 5

```
Unesite broj >> 32
Ima još brojeva (da ili ne)? da
Unesite broj >> 45
Ima još brojeva (da ili ne)? da
Unesite broj >> 34
Ima još brojeva (da ili ne)? da
Unesite broj >> 76
Ima još brojeva (da ili ne)? da
Unesite broj >> 45
Ima još brojeva (da ili ne)? jok
```

Prosek je 46.4

Sentinel petlja

- **sentinel petlja** obrađuje podatke sve dok ne naiđe na specijalnu vrednost koja označava kraj
- ta specijalna vrednost zove se **sentinel**
- sentinel se mora razlikovati od „običnih“ podataka jer se on ne obrađuje

Sentinel petlja 2

```
uzmi prvi podatak
while podatak nije sentinel
    obradi podatak
    uzmi naredni podatak
```

- prvi podatak se izdvoji pre nego što petlja počne – „priming read“
- ako je prvi podatak baš sentinel, petlja se preskače i nema obrade
- u suprotnom, podatak se obrađuje i čita se sledeći podatak

Sentinel petlja ₃

- recimo da računamo prosek pozitivnih brojeva
- neće biti broja manjeg od 0 – negativan broj će biti sentinel

Sentinel i računanje proseka

```
# average3.py

def main():
    sum = 0.0
    count = 0
    x = eval(raw_input(
        "Unesite broj (negativan za kraj) >> "))
    while x >= 0:
        sum = sum + x
        count = count + 1
        x = eval(raw_input(
            "Unesite broj (negativan za kraj) >> "))
    print "\nProsek je", sum / count
```

Sentinel i računanje proseka ₂

```
Unesite broj (negativan za kraj) >> 32
Unesite broj (negativan za kraj) >> 45
Unesite broj (negativan za kraj) >> 34
Unesite broj (negativan za kraj) >> 76
Unesite broj (negativan za kraj) >> 45
Unesite broj (negativan za kraj) >> -1
```

Prosek je 46.4

Sentinel i računanje proseka ₃

- ova verzija programa je jednostavna za korišćenje kao i prethodna (sa interaktivnom petljom)
- ali ne gnjavi korisnika sa unošenjem „da“ svaki put
- mana je što ne možemo raditi sa negativnim brojevima
- tada sentinel ne bi mogao biti broj

Sentinel i računanje proseka 4

- mogli bismo unositi sve podatke kao stringove
- ispravan unos mora se konvertovati u broj
- sentinel bi mogao da bude prazan string – ""

Prazan string kao sentinel

```
inicijalizuj sum na 0.0
inicijalizuj count na 0
unesi podatak kao string, xStr
while xStr != ""
    konvertuj xStr u broj x
    dodaj x na sum
    dodaj 1 na count
    unesi sledeći podatak kao string, xStr
ispiši sum / count
```

Prazan string kao sentinel 2

```
# average4.py

def main():
    sum = 0.0
    count = 0
    xStr = raw_input(
        "Unesite broj (<Enter> za kraj) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = raw_input(
            "Unesite broj (<Enter> za kraj) >> ")
    print "\nProsek je", sum / count
```

Prazan string kao sentinel ₃

```
Unesite broj (<Enter> za kraj) >> 34
Unesite broj (<Enter> za kraj) >> 23
Unesite broj (<Enter> za kraj) >> 0
Unesite broj (<Enter> za kraj) >> -25
Unesite broj (<Enter> za kraj) >> -34.4
Unesite broj (<Enter> za kraj) >> 22.7
Unesite broj (<Enter> za kraj) >>
```

Prosek je 3.38333333333

Petlje i fajlovi

- naš program je interaktivan – to može biti nezgodno
- šta ako je korisnik pogrešio kod unosa 43. broja od njih 50?
- kod veće količine podataka bolje je čitati podatke iz fajla

Čitanje iz fajla u petlji

```
# average5.py

def main():
    fileName = raw_input("Unesite ime fajla >> ")
    infile = open(fileName, 'r')
    sum = 0.0
    count = 0
    for line in infile.readlines():
        sum = sum + eval(line)
        count = count + 1
    print "\nProsek je", sum / count
```

Fajlovi i sentinel

- mnogi jezici nemaju mogućnost za čitanje iz fajla pomoću for petlje
- tada je potreban sentinel
- možemo da koristimo `readline` u petlji da čitamo red-po-red iz fajla
- na kraju fajla `readline` će vratiti prazan string ""
 - šta ako u fajlu postoji prazan red?

Fajlovi i sentinel 2

```
line = infile.readline()
while line != ""
    # obradi line
    line = infile.readline()
```

- da li ovaj kod pravilno radi u slučaju kada postoji prazan red u fajlu?
- za prazan red `readline` vraća `"\n"`

Fajlovi i sentinel 2

```
# average6.py

def main():
    fileName = raw_input("Unesite ime fajla >> ")
    infile = open(fileName, 'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        sum = sum + eval(line)
        count = count + 1
        line = infile.readline()
    print "\nProsek je", sum / count
```

Ugnježdene petlje

- videli smo kako je moguće ugnježdavati `if` naredbe
- na sličan način je moguće ugnježdavati i petlje
- recimo da u fajlu sa brojevima, u jednom redu može biti više brojeva razdvojenih zarezom

Ugnježdene petlje 2

- na najvišem nivou koristićemo petlju za obradu fajla koja računa sum i count

```
sum = 0.0
count = 0
line = infile.readline()
while line != "":
    # ažuriraj sum i count
    line = infile.readline()
print "\nProsek je", sum/count
```

Ugnježdene petlje 3

- na sledećem nivou treba ažurirati `sum` i `count`
- pošto svaki red u fajlu sadrži više brojeva razdvojenih zarezom, možemo taj string podeliti na podstringove
- svaki od podstringova će biti broj
- u petlji idemo kroz sve podstringove,
 - konvertujemo ih u broj
 - i dodajemo na `sum`
 - i inkrementiramo `count`

Ugnježdene petlje 4

```
for xStr in line.split(","):
    sum = sum + eval(xStr)
    count = count + 1
```

- ova for petlja koristi line, što je promenljiva iz spoljne petlje

Ugnježdene petlje 5

```
# average7.py

import string

def main():
    fileName = raw_input("Unesite ime fajla >> ")
    infile = open(fileName, 'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        for xStr in line.split(","):
            sum = sum + eval(xStr)
            count = count + 1
        line = infile.readline()
    print "\nProsek je", sum / count
```

Ugnježdene petlje ₆

- petlja koja obrađuje brojeve u jednom redu je uvučena ispod petlje koja čita redove iz fajla
- spoljašnja while petlja iterira jednom za svaki red u fajlu
- za svaki ciklus spoljašnje petlje, unutrašnja petlja iterira sve cikluse (zavisno od broja brojeva u tom redu)
- kada se završi unutrašnja petlja, čita se sledeći red i proces se ponavlja

Logički izrazi

- `if` i `while` koriste logičke (Bulove) izraze
- vrednost logičkog izraza može biti `True` ili `False`
- do sada smo pisali samo izraze poređenja (`while x>0:`)

Logički operatori

- ovakvi jednostavni izrazi nekad nisu dovoljni
- recimo da treba odrediti da su dve 2D tačke na istom mestu
- njihove x koordinate moraju biti jednake
- i y koordinate moraju biti jednake

Logički operatori 2

```
if x1 == x2:  
    if y1 == y2:  
        # tačke su jednake  
    else:  
        # tačke nisu jednake  
else:  
    # tačke nisu jednake
```

- ovakav test izgleda rogobatno
- možemo koristiti logičke operatore and, or i not

Operatori and i or

- operatori and i or se koriste za kombinovanje dva logička izraza i daju logički rezultat
- očekuju dva **operanda**: zovemo ih **binarni** operatori

<izraz1> and <izraz2>

<izraz1> or <izraz2>

Operator and

- operator and predstavlja konjunkciju dva logička izraza
- vraća True samo kad su **oba** izraza True

P	Q	P and Q
T	T	T
T	F	F
F	T	F
F	F	F

- P i Q su operandi
- postoji 4 moguće kombinacije vrednosti

Operator or

- operator or predstavlja disjunkciju dva logička izraza
- vraća True ako je bilo koji od izraza True

P	Q	P or Q
T	T	T
T	F	T
F	T	T
F	F	F

- or vraća False samo kad su oba operanda False
- or vraća True kad su oba operanda True – to se razlikuje od korišćenja reči „ili“ u prirodnom jeziku!

Operator not

- operator not predstavlja negaciju logičkog izraza
- not je **unarni** operator – ima jedan operand

P	not P
T	F
F	T

Kombinovanje logičkih operatora

- operatore možemo kombinovati i tako graditi složene izraze
- izračunavanje takvih izraza zavisi od **prioriteta operatora**
- prioritet operatora je u opadajućem redosledu: not, and, or
- na primer:
a or not b and c
- je isto što i
(a or ((not b) and c))
- treba koristiti zagrade da se izbegne zabuna

Kombinovanje logičkih operatora 2

- za poređenje 2D tačaka možemo koristiti `and`

```
if x1 == x2 and y1 == y2:  
    # tačke su jednake  
else:  
    # tačke nisu jednake
```

- ceo izraz će biti `True` samo kada su **oba** manja izraza `True`

Racquetball

- sport sa reketom i lopticom u zatvorenom prostoru, slično skvošu
- igra se dok jedan od igrača ne osvoji 15 poena
`scoreA == 15 or scoreB == 15`
- kada je bar jedan od izraza `True`, ceo izraz je `True`
- treba nam petlja koja radi sve dok igra **nije završena**
- možemo negirati gornji izraz:

```
while not(scoreA == 15 or scoreB == 15):  
    # nastavi da igraš
```

Racquetball 2

- ako jedan igrač osvoji 7 poena dok drugi ne osvoji nijedan, igra se završava

```
while not(scoreA == 15 or scoreB == 15 or
(scoreA == 7 and scoreB == 0) or
(scoreB == 7 and scoreA == 0):
    # nastavi da igraš
```

Odbojka

- jedan set se igra do 25 poena
- set se mora dobiti sa 2 poena razlike

$(a \geq 25 \text{ and } a-b \geq 2) \text{ or } (b \geq 25 \text{ and } b-a \geq 2)$

$(a \geq 25 \text{ or } b \geq 25) \text{ and } \text{abs}(a-b) \geq 2$

Bulova algebra

- veština pisanja logičkih izraza je veoma važna
- logički izrazi poštuju zakone **Bulove algebre**

algebra	Bulova algebra
$a \cdot 0 = 0$	<code>a and False == False</code>
$a \cdot 1 = a$	<code>a and True == a</code>
$a + 0 = a$	<code>a or False == a</code>

- `and` liči na množenje
- `or` liči na sabiranje
- `0` liči na `False`, `1` liči na `True`

Bulova algebra ₂

- bilo šta or-ovano sa True je True
 $a \text{ or } \text{True} == \text{True}$
- distributivnost and i or
 $a \text{ or } (b \text{ and } c) == (a \text{ or } b) \text{ and } (a \text{ or } c)$
 $a \text{ and } (b \text{ or } c) == (a \text{ and } b) \text{ or } (a \text{ and } c)$
- dvostruka negacija se poništava
 $\text{not}(\text{not } a) == a$
- De Morganovi zakoni
 $\text{not}(a \text{ or } b) == (\text{not } a) \text{ and } (\text{not } b)$
 $\text{not}(a \text{ and } b) == (\text{not } a) \text{ or } (\text{not } b)$

Bulova algebra ₃

- na osnovu ovih pravila možemo pojednostaviti neke izraze

```
while not(scoreA == 15 or scoreB == 15):
    # nastavi da igraš
```

- primenom De Morganovog zakona

```
while (not scoreA == 15) and (not scoreB == 15):
    # nastavi da igraš
```

- što se svodi na

```
while scoreA != 15 and scoreB != 15:
    # nastavi da igraš
```

Bulova algebra ₄

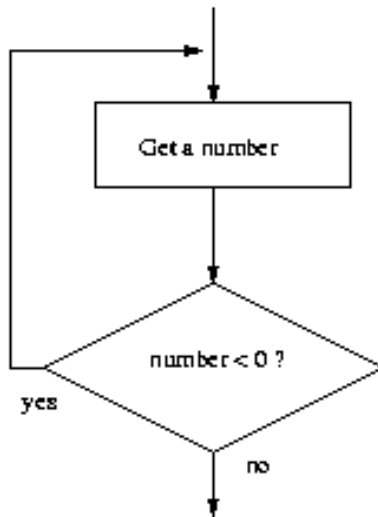
- nekad je lakše formulisati uslov kada petlja treba da se završi, nego kada da nastavi sa radom
- samo dodamo `not` na početak takvog izraza
- pomoću De Morganovih zakona možemo dalje pojednostaviti izraz

Petlja sa izlaskom na dnu

- for i while mogu da posluže da izrazimo bilo koji algoritam
- međutim, nekada je čitljivije koristiti petlje sa izlaskom na dnu (umesto na vrhu)
- primer:

```
repeat
    unesi broj
until number >= 0
```

Petlja sa izlaskom na dnu 2



Petlja sa izlaskom na dnu ₃

- telo petlje sa izlaskom na dnu izvrši se **obavezno** bar jednom
- Python nema naredbu za ovakve petlje ali se ona može napraviti pomoću `while`
- pripremimo vrednosti tako da uslov bude uvek ispunjen pre prvog ciklusa petlje

```
number = -1
while number < 0:
    # uradi nešto
```

Petlja sa izlaskom na dnu 4

- u Pythonu se ovakva petlja može simulirati i pomoću break naredbe
- break služi za momentalno iskakanje iz petlje
- može i za iskakanje iz beskonačne petlje

```
while True:
```

```
    number = eval(raw_input("Unesite broj >> "))
```

```
    if number >= 0:
```

```
        break # izadi iz petlje ako je broj ispravan
```

- bilo bi lepo da program ispiše upozorenje da broj nije ispravan

Petlja sa izlaskom na dnu 5

- u verziji sa while petljom ovo je rogovatno

```
number = -1
while number < 0:
    number = eval(raw_input("Unesite pozitivan broj: "))
    if number < 0:
        print "Uneti broj nije pozitivan!"
```

- vršimo proveru na dva mesta – nije elegantno

Petlja sa izlaskom na dnu 6

- verzija sa break samo još dodaje else klauzulu

```
while True:
    number = eval(raw_input("Unesite pozitivan broj: "))
    if number >= 0:
        break
    else:
        print "Uneti broj nije pozitivan!"
```

Petlja ipo

- nešto elegantnija verzija

```
while True:
    number = eval(raw_input("Unesite pozitivan broj: "))
    if number >= 0:
        break
    print "Uneti broj nije pozitivan!"
```

- izlaz iz petlje je u sredini tela petlje
- → „petlja ipo“

Petlja ipo

- petlja ipo je elegantan način da se izbegne inicijalno čitanje u sentinel petlji

```
while True:
```

```
    uzmi sledeći podatak
```

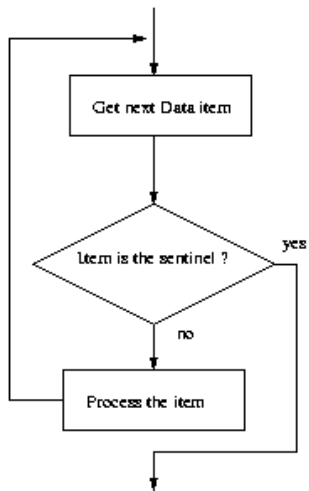
```
    if podatak je sentinel:
```

```
        break
```

```
    obradi podatak
```

- sentinel vrednost se ne obrađuje
- ne moramo inicijalizovati ništa pre petlje

Petlja ipo



Koristiti break: da ili ne?

- korišćenje `break` je stvar ukusa
- zbog lakšeg čitanja koda, `break` treba koristiti samo kad je neophodno
- teško je pratiti kod u kome ima više `break`-ova za izlazak iz petlje

Logički izrazi i grananje

- logički izrazi se mogu koristiti za kontrolu toka programa
- recimo da pišemo program koji se izvršava sve dok korisnik unosi odgovor koji počinje sa "y"
- jedno rešenje:

```
while response[0] == "y" or response[0] == "Y":
```

Logički izrazi i grananje 2

- samo pažljivo! ne možemo pisati „skraćeno“:
`while response[0] == "y" or "Y":`
- zašto ovo ne radi?
- Python interno predstavlja `bool` tip pomoću brojeva 1 (za `True`) i 0 (za `False`)
- relacioni operatori kao `==` uvek vraćaju `bool` vrednost

Logički izrazi i grananje 3

- Python će dopustiti da se svaki drugi tip tretira kao bool
- za brojeve: 0 se smatra za False, sve ostalo je True
- za liste: prazna lista je False, neprazna je True
- za stringove: prazan string je False, neprazan je True

```
>>> bool(0)
False
>>> bool(1)
True
>>> bool(32)
True
>>> bool("Hello")
True
>>> bool("")
False
>>> bool([1,2,3])
True
>>> bool([])
False
```


Logički operatori i drugi tipovi podataka

- logičke operatore možemo primeniti i na druge tipove podataka, ne samo `bool`
- značenje operatora u tom slučaju je sledeće:

operator	značenje
<code>x and y</code>	ako je <code>x</code> <code>False</code> vrati <code>x</code> , inače vrati <code>y</code>
<code>x or y</code>	ako je <code>x</code> <code>True</code> vrati <code>x</code> , inače vrati <code>y</code>
<code>not x</code>	ako je <code>x</code> <code>False</code> vrati <code>True</code> , inače vrati <code>False</code>

Operator and i drugi tipovi podataka

- razmotrimo izraz `x and y`
- ako je `x True`, tada vrednost celog izraza zavisi od `y`
- vraćanjem `y`, ako je `y True`, `True` se i vraća
- ako je `y False`, `False` se i vraća

Izračunavanje logičkih izraza

- prilikom izračunavanja logičkog izraza `True` ili `False` se vraćaju čim je rezultat poznat
- ne mora se ceo izraz izračunati!
- u `and`-izrazu, ako je prvi operand `False`, rezultat je obavezno `False`
- u `or`-izrazu, ako je prvi operand `True`, rezultat je obavezno `True`

Izračunavanje logičkih izraza: primer

- pogledajmo izraz
`response[0] == "y" or "Y"`
- on je isto što i
`(response[0] == "y") or ("Y")`
- da bi ovaj izraz bio True:
 - treba da je `response[0] == "y"`
 - "Y" je neprazan string – tretira se kao True
- ovaj izraz će uvek biti True, bez obzira šta korisnik unese!

Izračunavanje logičkih izraza: primer 2

```
ans = input("Koji sladoled želite [vanila]: ")
if ans:
    flavor = ans
else:
    flavor = "vanila"
```

- ako korisnik samo pritisne Enter, ans će biti prazan string
- prazan string se tretira kao False

Izračunavanje logičkih izraza: primer 2 ₂

- ovo može još sažetije!

```
ans = input("Koji sladoled želite [vanila]: ")  
flavor = ans or "vanila"
```

- ili još kraće:

```
flavor = input("Koji sladoled želite [vanila]: ") or "vanila"
```

Izračunavanje logičkih izraza

- ovakav kod može biti teško razumljiv
- zbog čitljivosti – treba se uzdržati od prekomerne upotrebe ovih „trikova“