09a oblikovanje nizov

January 28, 2024

1 Oblikovanje nizov

Pri izpisovanju smo bili doslej omejeni na dokaj zoprni `print`. Za začetek nam je zadoščal, kmalu pa postane neroden, saj nekaterih stvari z njim preprosto ni mogoče (preprosto) narediti. Za začetek se je potrebno z njim pogajati, da ne gre po vsakem izpisu v novo vrsto in da ne dodaja presledkov med reči, ki jih izpisujemo. Mu je mogoče reči, naj pri izpisu števil ne izpiše vseh šestnajst ali kolikor že decimalk? Tudi ne. `print` se s takimi rečmi ne ukvarja.

[1]:
```
print(6 / 7)
```

```
0.8571428571428571
```

Kaj pa, če bi želeli število izpisati na samo dve decimalki? Res, nerodno.

Večina besedil, ki jih izpišejo ali kako drugače pokažejo programi, vsebuje fiksno besedilo, v katerega so vstavljeni kaki nizi ali številke. Če računalnik izpiše "Datoteke kraljice.py ni mogoče najti", je vstavljeno besedilo "kraljice.py", ki je najbrž vsakič drugačno, odvisno pač od tega, katero datoteko (neuspešno) iščemo. V "60 Fahrenheitov je 16 Celzijevih stopinj" sta vstavljeni števili 60 in 16, ki sta odvisni od tega, v katerem semestru poženemo program, ostalo besedilo pa je vedno enako.

Večina računalniških jezikov omogoča, da sestavimo nize tako, da napišemo "konstantni" niz, označimo mesta, na katera je potrebno nekaj vstaviti. Potem pa na nek način povemo, kaj naj vstavi. Popularni prevajani jeziki so večinoma povzeli način, ki so si ga izmislili v jeziku C; prepoznali ga boste po `%i %5.3f` in podobnih rečeh. Novejši jeziki, predvsem skriptni, ki so bližje spletu, pa navadno omogočajo preprostejše in zmogljivejše načine sestavljanja nizov iz vzorcev.

Eno od načel Pythona je, da se da vsako stvar narediti na en in samo en očiten način. To je dobro zato, ker, posledično, vsi programerji pišemo podobne programe in zato lažje sodelujemo. Prav pri oblikovanju nizov pa se je to sfižilo: nize lahko oblikujemo na tri različne načine. Do tega je prišlo, ko je Python z leti postajal pametnejši. V začetku je uporabljal nekoliko izboljšano različico C-jevskega oblikovanja, saj druge praktično ni bilo. Kasneje so dodali veliko preprostejši način oblikovanja nizov, C-jevskega pa obdržali zato, da so stari programi še vedno delovali. Leta so tekla in programski jeziki napredovali. V modo je prišel drugačen še veliko prikladnejši način in Python je bil s svojima, precej nerodnejšima, videti prav štorast. Zato lahko od različice 3.6 naprej tudi v Pythonu oblikujemo nize na enak način kot v drugih modernih jezikih.

Pri tem predmetu se bomo naučili le tretji načinu. Prva dva je potrebno poznati, če boste resno programirali v Pythonu in boste morali brati tudi tuje programe. Sami pa ju boste rekdo uporabili.

1.1 f-nizi

Recimo, da imamo

[2]:
```
fahr = 42
celz = (fahr - 32) * 5 / 9
```

Zdaj bi radi sestavili, v bistvu, takšen niz:

[3]:
```
"{fahr} Fahrenheitov je {celz} Celzijev"
```

[3]:
```
'{fahr} Fahrenheitov je {celz} Celzijev'
```

le da želimo, da Python na mesti, ki smo ju označili s `{fahr}` in `{celz}`, vstavi vrednosti spremenljivk `fahr` in `celz`.

To počnejo f-nizi. F-niz je kot običajen niz, le da mu pred prvi narekovaj dodamo črko f (kot *format*). V takšnih nizih imajo zaviti oklepaji poseben pomen: kar je v njih, se (izračuna in) vstavi.

[4]:
```
f"{fahr} Fahrenheitov je {celz} Celzijev"
```

[4]:
```
'42 Fahrenheitov je 5.555555555555555 Celzijev'
```

V neki domači nalogi smo - oh, kako je bilo to nerodno! - pisali

```
from random import randint

a = randint(2, 10)
b = randint(2, 10)
print(a, "krat", b)
c = input("Odgovor? ")
```

S tem, da najprej izpišemo in potem zahtevmo odgovor, smo se izognili temu, da bi morali mukoma sestavljali niz ob klicu funkcije `input`:

```
a = randint(2, 10)
b = randint(2, 10)
c = input("Koliko je " + str(a) + "x" + str(b) + "?")
```

Zdaj, ko poznamo metodo f-nize, poznamo preprostejši način:

```
a = randint(2, 10)
b = randint(2, 10)
c = input(f"Koliko je {a}x{b}? ")
```

(Mimogrede omenimo starejša načina oblikovanja nizov, ki smo ju omenjali na začetku: v C-jevskem slogu bi pisali `c = input("Koliko je %sx%s? " % (a, b))`, v onem malo novejšem pa `c = input("Koliko je {}x{}? ".format(a, b))`. Obema je skupno to, da so v nizu le označena mesta, kamor je potrebno vstavljati, potem pa moramo na nekem drugem mestu, po koncu niza, našteti stvari, ki jih vstavljamo. Z novejšim slogom lahko z nekimi čudnimi triki v zavite oklepaje dodajamo tudi imena spremenljivk, vendar niti približno tako udobno kot po novem.)

Na ta način ne vstavljamo le števil, temveč karkoli, kar se da izpisati.

[5]:
```
datoteka = "kraljice.py"
f"Datoteke {datoteka} ne najdem"
```

[5]:
```
'Datoteke kraljice.py ne najdem'
```

1.2 Izrazi v f-nizih

V nize ne vstavljamo le vrednosti spremenljivk: v zavite oklepaje lahko pišemo kar cele izraze.

[6]:
```
fahr = 42

f"{fahr} Fahrenheitov je {(fahr - 32) * 5 / 9} Celzijev"
```

[6]:
```
'42 Fahrenheitov je 5.555555555555555 Celzijev'
```

1.3 Kako oblikujemo reči

Nize zdaj znamo zlagati, tistega, kar nas je najbolj motilo, pa se sploh še nismo lotili: 60.0 Fahrenheitov je 15.555555555555555 Celzijev? Za moje potrebe bi popolnoma zadostovalo 15.5 in tudi pri 15 me ne bi nič bolj zeblo.

Če bi radi povedali, kako naj se izpiše neko število (ali niz ali karkoli že), dodamo izrazu dvopičje in za njim opis formata.

To bomo najpogosteje potrebovali za oblikovanje necelih števil. To gre tako:

[7]:
```
f"{fahr:4.1f} Fahrenheitov je {celz:4.1f} Celzijev"
```

[7]:
```
'42.0 Fahrenheitov je  5.6 Celzijev'
```

Opis formata je tule `4.1f`. Pri tem `4.1` pomeni, da bi radi izpis na štiri mesta, pri čemer naj bo eno rezervirano za decimalko. Za črko `f` si predstavljajte, da pomeni `float`.

Če pustimo za število premalo prostora, ga bo izpis pač zasedel več.

[8]:
```
x = 1234.5678
f"Primer predolgega števila: {x:3.1f}"
```

[8]:
```
'Primer predolgega števila: 1234.6'
```

Zahtevali smo eno decimalko in to smo tudi dobili, vse skupaj pa je širše kot štiri mesta, saj je število pač predolgo.

Določanje števila decimalk je smiselno le pri necelih številih. Če vstavljamo nize ali cela števila, lahko določamo le širino:

[9]:
```
podatki = [
    (74, "Anze", False),
    (82, "Benjamin", False),
```

```
    (48, "Cilka", True),
    (66, "Dani", False),
    (61, "Eva", True),
    (101, "Franc", False),
    ]

for teza, ime, spol in podatki:
    print(f"{ime:10}{teza:6}")
```

```
Anze          74
Benjamin      82
Cilka         48
Dani          66
Eva           61
Franc        101
```

Tu za številom mest nismo dodajali črke `f`, saj ne gre za (necela) števila. Za nizi pa sploh ne. (Obstajajo sicer druge črke, ki bi jih lahko dodali, da Python povemo, za kaj gre, vendar jih smemo tule brez škode pozabiti.)

Vsa imena so izpisana z desetimi znaki; manjkajoči prostor je zapolnjen s presledki. Številke so izpisane s šestimi mesti, manjkajoči prostor spet zapolnjujejo presledki.

Nizi so poravnani na levo - presledki so dodani za njimi. Večina reči, ki jih izpisujemo (nizi in števila namreč niso edino, kar se da izpisati), so poravnani tako. Številke pa so poravnane desno. To lahko spremenimo, če za dvopičje dodamo `<`, `>` ali `^`, s katerimi naročimo, naj bo reč poravnana levo, desno ali na sredino. Ravno obratni izpis kot zgoraj bi dosegli z

[10]:
```
for teza, ime, spol in podatki:
    print(f"{ime:>10}{teza:<6}")
```

```
      Anze74
  Benjamin82
     Cilka48
      Dani66
       Eva61
     Franc101
```

To ni ravno posrečeno. Dodajmo še presledek.

[11]:
```
for teza, ime, spol in podatki:
    print(f"{ime:>10} {teza:<6}")
```

```
      Anze 74
  Benjamin 82
     Cilka 48
      Dani 66
       Eva 61
     Franc 101
```

To je lažje berljivo, še vedno pa je seveda najlepši prvi izpis.

Pred znak, ki določa poravnavo, lahko dodamo simbol, ki naj se uporabi za zapolnjevanje - če seveda nismo zadovoljni s presledkom. Poravnajmo imena levo, števile desno, namesto presledkov pa zapolnimo prazen prostor s pikami.

[12]:
```
for teza, ime, spol in podatki:
    print(f"{ime:.<10}{teza:.>6}")
```

```
Anze…74
Benjamin…82
Cilka…48
Dani…66
Eva…61
Franc…101
```

Namesto pik lahko uporabimo poljuben drug znak. Kadar takole zapolnjujemo prostor s čimerkoli drugim kot s presledki, moramo dodati znake za poravnavanje (`<`, `>` ali `^`), tudi kadar z njimi izberemo takšno poravnavanje, kot bi bilo tudi privzeto (desno za števila, levo za vse drugo) všeč.

V vse detajle ne bomo šli. Določiti je mogoče še veliko reči. Pri številih lahko zahtevamo, naj se vedno izpiše predznak, torej +, kadar je število pozitivno. Za števila lahko določimo, naj se izpišejo dvojiško ali šestnajstiško… Kdor potrebuje več kot to, naj Googla.

1.4 Kontrolni znaki v nizih

Niz se lahko razprostira prek več vrstic, če ga zapremo v trojne narekovaje.

[13]:
```
s = """Niz, ki je
dolg vec vrstic.
Konkretno, tri!"""

print(s)
```

```
Niz, ki je
dolg vec vrstic.
Konkretno, tri!
```

V nizu so očitno shranjena mesta, ko mora pri izpisu iti v novo vrstico. Kako?

Poleg "običajnih" znakov, ki jih je mogoče izpisati, vsebuje niz tudi kontrolne znake. Če jih želimo "videti", to lahko storimo tako, da niz izpišemo v ukazni vrstici, vendar brez `print`.

[14]:
```
s
```

[14]:
```
'Niz, ki je\ndolg vec vrstic.\nKonkretno, tri!'
```

Znak `\n` ni pravi znak (in, predvsem, nista dva znaka, temveč en sam) in pomeni prehod v novo vrstico. Napišemo ga lahko tudi sami.

[15]:
```
s = "Tole gre pa\nv dve vrstici"
```

Se pravi, vedno, ko v niz napišemo `\n`, Python tega ne bo razumel kot vzvratno poševnico (*backslash*), temveč kot znak za prehod v novo vrsto (*new line*). Podobnih znakov je še več, vendar navadno uporabljamo le še enega, namreč `\t`, ki pomeni tabulator. Pa ga preskusimo.

[16]:
```
for i in range(5, 15):
    print(f"{i}\t{i**2}")
```

```
5       25
6       36
7       49
8       64
9       81
10      100
11      121
12      144
13      169
14      196
```

Zaporedjem, ko sta `\n` in `\t` pravimo ubežna zaporedja (*escape sequence*).

V računalnikovem pomnilniku so znaki shranjeni kot številke: vsakemu znaku ustreza določena številka. V kodiranju ASCII, ki ga običajno uporabljamo, je, npr. koda znaka A 65, koda B je 66, koda C je 67 in tako naprej. Kode a, b in c so 97, 98 in 99… Koda presledka je 32. Vsi kontrolni znaki imajo kode manjše od 32. Tako je koda znaka `\n` 10 in koda tabulatorja, `\t` je 9. Več o ASCII si lahko preberete tule. Na tej strani se nahaja tudi seznam kod ASCII.

Da gre pri `\n` za en znak in ne za dva, se lahko hitro prepričamo.

[17]:
```
s = "a\nb"
print(s)
```

```
a
b
```

[18]:
```
print(len(s))
```

```
3
```

[19]:
```
s[1]
```

[19]:
```
'\n'
```

Znake lahko opisujemo tudi z njihovimi kodami. To storimo z ubežnim zaporedjem `\x`, ki ji sledi koda znaka, zapisana v šestnajstiškem številskem sistemu. Tako, je, npr. koda črke a 97, to je 61 po šestnajstiško, koda z pa 122 oz. 7a po šestnajstiško. Zato lahko niz "Janez" napišemo tudi takole.

[20]:
```
"J\x61ne\x7a"
```

[20]:
```
'Janez'
```

Tega seveda ne počnemo, ker nima smisla. Pač pa nam ubežno zaporedje pride prav, kadar želimo zapisovati kake nenavadne kontrolne znake. Včasih bomo naleteli, na primer, na znak s kodo 0. Tega zapišemo kot `\x00` (pazite, napisati je potrebno dve ničli!).

Vzvratna poševnica (*backslash*) ima torej v nizih posebno vlogo: uporablja se za ubežno zaporedje. Kaj pa, kadar želimo, da bi bila vzvratna poševnica samo vzvratna poševnica? V tem primeru naredimo dve. Torej: `\\` je ubežno zaporedje, ki predstavlja vzvratno poševnico (eno samo!). Kupiš dva, dobiš enega. Na Windowsih bomo to reč najpogosteje srečali v imenih direktorijev.

[21]:
```
dir = "c:\\documents and settings\\nina\\telovadba"
print(dir)
```

```
c:\documents and settings\nina\telovadba
```

Še enkrat, ker je pomembno: ko pišemo imena direktorijev, vedno pišemo dvojne vzvratne poševnice, da bomo dobili enojne. Ali pa, še boljše: pišimo kar običajne poševnice, enojne. Tako napisani programi bodo delovali tudi na Linuxu in Macu.

Če pozabimo pisati dvojne poševnice, dobimo, kar si zaslužimo, kar ni nujno tisto, kar bi radi.

[22]:
```
dir = "c:\documents and settings\nina\telovadba"
print(dir)
```

```
c:\documents and settings
ina     elovadba
```

`\n` in `\t` sta znaka za novo vrstico in za tabulator.

Pisanju dvojnih poševnic se lahko izognemo z r-nizi (*r* kor *raw*). Ti so podobni f-nizom, le da pred narekovaj damo črko `r` in vzvratna poševnica bo samo vzvratna poševnica.

[23]:
```
dir = r"c:\documents and settings\nina\telovadba"
print(dir)
```

```
c:\documents and settings\nina\telovadba
```

V tem primeru v niz ne moremo dati ubežnega zaporedja `\n`, saj bosta `\` in `n` v tem primeru res pomenila samo `\` in `n`.

Med vsemi drugimi ubežnimi zaporedji omenimo le še dve: `\'` in `\"` predstavljata enojni in dvojni narekovaj.

Mimogrede, če vam bo kdaj prišlo prav: kodo znaka dobimo s funkcijo `ord`.

[24]:
```
ord("A")
```

[24]: 65

[25]:
```
ord("B")
```

[25]: 66

[26]:
```
ord("a")
```

[26]: 97

[27]:
```
ord("\n")
```

[27]: 10

V drugo smer vodi funkcija `chr`, ki ji povemo kodo znaka in vrne niz, ki vsebuje znak s to kodo.

[28]:
```
chr(65)
```

[28]: 'A'

[29]:
```
chr(66)
```

[29]: 'B'

[30]:
```
chr(10)
```

[30]: '\n'