03a Kako racunalnik shrani besedilo

January 28, 2024

0.1 Kako je shranjeno besedilo

Računalnik v resnici shranjuje samo številke. Tako v pomnilniku (ko imamo `str`) kot na disku (v datotekah) so samo številke. Tudi, ko znak potuje po internetu, potuje kot številka. Vse te številke se spremenijo v znake šele ob izpisu.

Nekoč se je bilo torej potrebno dogovoriti, s kakšno številko bo predstavljen kateri znak. V onih davnih dneh še ni bilo računalnikov in nihče si ni predstavljal interneta in vseh številnih pisav in, uh, celo emojijev, ki jih bo potrebno nekako shranjevati v pomnilnik, datoteke in pošiljati po mrežnih povezavah. Standardov je bilo zato več, neenotnih, za različne namene. Od vseh je preživel bolj ali manj samo ASCII. Ta pravi, recimo, da veliki A predstavimo s številko 65, B s 66, C z 67… Mali a je 97, b 98 … in tako naprej. Svoje kode imajo tudi drugi znaki: presledek je 32, pika je 46, oklepaj 40…

O tem se zlahka prepričamo: funkcija `ord` sprejme znak in vrne kodo tega znaka.

[1]:
```
ord("A")
```

[1]:
```
65
```

[2]:
```
ord("a")
```

[2]:
```
97
```

[3]:
```
ord(" ")
```

[3]:
```
32
```

[4]:
```
ord(".")
```

[4]:
```
46
```

To funkcijo boste redko potrebovali. Nič ne bo narobe (in pravzaprav bo morda celo koristno), če jo pozabite. Vseeno povem še za obratno funkcijo: funkciji `chr` podamo kodo in vrne znak.

[5]:
```
chr(65)
```

[5]:
```
'A'
```

Problem 1: ASCII je bil razvit za teleprinterje. Ti so bili videti kot nekakšni pisalni stroji na daljavo: kar je nekdo tipkal, se je prek žice ali antene preneslo na drugo stran Amerike, kjer je teleprinter to izpisoval. Teleprinterji so imeli tudi funkcije, kot je "izvrzi list in vzemi naslednjega" ali "zacingljaj" in podobno. Tudi te funkcije so zahtevale svojo številko. Če je teleprinter, recimo, prejel številko 12, je vzel naslednji list papirja. Tem rečem rečemo *kontrolne kode*. O njih več v naslednjem razdelku. Omenjamo jih zgolj zaradi …

Problem 2: ASCII je (bil) sedembitni standard, torej je na voljo samo 128 številk. Pri tem so kar 32 številk zasedle kontrolne kode. Preostalih 96 znakov je zadoščalo za angleško abecedo, ločila in še par malenkosti, za trde in mehke č-je pa se Američani, ki so sestavljali standard, niso posebej menili. Kitajci pa si niti s celotnim naborom 96 mest ne bi mogli veliko pomagati.

ASCII je doživel kup razširitev. Že v sedembitni različici smo ga v rajnki domovini popravili v YUSCII, v katerem smo žrtvovali nekaj znakov, kot so zaviti oklepaji in vzvratne poševnice in jih zamenjali s šumniki. Microsoft in IBM sta kasneje sestavila vsak svoj standard, takoimenovane kodne tabele; gre za razporede, v katerih je "spodnji del", znaki s kodami do 127, enak kot v izvirnem ASCII, v gornjem delu - kode od 128 do 256 - pa so različni drugi znaki. V naših krajih se je najbolj prijela tabela CP1250, namenjeni srednje- in vzhodnoevropskim jezikom, v kateri ima, recimo, č kodo 232. V zahodnoevroski tabeli, CP1252, č-ja ni, na mestu 232 pa je znak è. Poleg teh, Microsoftovih tabel, obstajajo IBMove, kjer je Slovanom namenjen IBM 852; da bi bilo programiranje še preprostejše, obstaja tudi tretji standard, ISO-8859, ki nam je namenil tabelo ISO-8859-2, ki je podobna, vendar ne povsem enaka CP1250.

Ko torej računalnik vidi kakšne številke in jih mora prikazati kot besedilo, mora vedeti, v katerem kodnem naboru je zapisan, se pravi, kako razumeti, recimo, številko 232. Če je besedilo v CP1252, je to è, če v CP1250, pa č. In kako naj to ve? V splošnem ne more! Spletne strani navadno vsebujejo glavo, v kateri je zapisan podatek o uporabljenem kodnem naboru. Če tega ni, se bodo šumniki pokazali prav ali pa tudi ne. In še huje: vse besedilo je napisano v istem kodnem naboru, zato se lahko zgodi, da ne bo moglo vsebovati tako črke č kot è. (To morda razloži kaj v zvezi s podnapisi filmov, ki napačno sugerirajo, da so na pikniku jedle *èvapèièe s èebulo*?)

Obenem pa s tem Kitajci še vedno ne morejo biti zadovoljni, saj jim dodatnih 128 znakov bore malo pomaga.

Takole smo se hecali do pred nekaj leti, ko je prišel v širšo rabo standard Unicode. Ta je narejen takole: vsakemu znaku je prirejena ena koda. Kod je 232, kar je čez in čez dovolj tudi za Kitajce. Nekatere znake je mogoče pisati tudi na več načinov: obstaja, recimo, znak za strešico in č lahko zapišemo kot č (koda 269) ali kot strešico (309) in c (99).

Da bi lahko zapisal 232 kod, bi potrebovali po štiri bajte na znak. Američanov to ne bi posebej osrečilo, saj bi se dolžina njihovih besedil početverila in bi se spraševali, kaj je bilo pravzaprav narobe s starim dobrim ASCII. Zato so se snovalci standarda domislili različnih načinov, kako "stisniti" zapis, da bo zahteval manj kot štiri bajte na znak. Med njimi je daleč najbolj razširjen UTF-8, občasno pa vidimo še UTF-16. UTF-8 je sestavljen zelo zvito: če imamo besedilo v starem dobrem ASCIIju (brez neangleških znakov, torej brez kakšnih šumnikov ali francoskih naglasov), se lahko delamo, da je v UTF-8. Čim vsebuje šumnike (recimo po standardu CP1250), pa ni več združljiv z UTF-8 in moramo, če ga hočemo pravilno prebrati, vedeti, po katerem standardu je kodirano.

Smo prišli z dežja pod kap? Smo imeli prej polno kodnih naborov (CP1250 do CP125bogvekoliko, pa IBMovi nabori, pa ISO-8859), zdaj pa imamo kup UTFjev? Niti ne. Razlika je v tem, da so

stari nabori določali, kakšne kode imajo posamezni znaki. Po novem imajo vsi znaki določene kode, obstaja le par načinov zapisa teh kod. Med njimi pa vsaj na zahodu prevladuje UTF-8.

Če imamo torej neko besedilo - recimo v neki datoteki, ali pa na neki spletni strani -, ki ga želimo prebrati, bo ponavadi zapisano v UTF-8 ali CP1250. Če je angleško in vsebuje le angleške znake, je to eno in isto. Če gre za slovensko besedilo, pa moramo vedeti, kako je zapisano. Če gre za novejše besedilo, je verjetno v UTF-8, če je starejše kot kakih deset let ali pa prihaja iz vira, s katerim ima kaj opraviti kakšen domači amaterskih izdelovalec spletnih strani, pa CP-1250. CP-1250 se žal še vedno kar pogosto pojavlja tudi v bližini Windowsov. Vedno ga lahko tudi poskusimo najprej prebrati kot UTF-8; če to uspe, je najbrž v resnici UTF-8, sicer ga preberemo kot CP1250.

Kako Pythonu povemo, kako naj bere besedilo? Metoda `open` ima še par argumentov. Ta, ki nas zanima, se imenuje `encoding` in ga navadno podamo poimensko: namesto, da bi napisali zgolj `open("drazba.txt", "utf8")` (kar pravzaprav ne bi delovalo, ker način kodiranja ni drugi argument, temveč sta pred njim še dva druga), napišemo `open("dražba.txt", encoding="utf8")`.

Pa če Pythonu ne povemo, kot kaj želimo brati besedilo - kot kaj ga bo poskusil brati? To je odvisno od nastavitev sistema - zato imajo Windowsi nastavitev "System locale". Tam sprašuje po "Current language for non-Unicode programs"; če imate nastavljeno slovenščino, bo privzeti kodni nabor CP1250. Na Ubuntuju in Macu je kot privzeto kodiranje nastavljen UTF-8.

Tule bi se lahko pogovorili še marsikaj. Predvsem ne vsebujejo vse datoteke besedil. Vendar so te stvare že precej tehnične in nekatere tudi specifične za Python, tako da se tu morda ustavimo.

0.2 Kontrolni znaki v nizih

V začetku prejšnjega razdelka smo omenili kontrolne znake. Večina bi sodila na smetišče računalniške zgodovine, vendar, kaj hočemo, so kjer so in bodo tam ostali za vso večnost. Za nas je relevantnih le par.

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

[8]:
```
s = """Niz, ki je
dolg vec vrstic.
Konkretno, tri!"""

print(s)
```

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

To je en sam niz, ne trije. Torej mora očitno vsebovati tudi nek indikator za konec vrstice. Ker so, kot zdaj vemo, nizi le zaporedja številk, torej obstaja številka, ki pomeni konec vrstice. Konkretno, gre za številko 10. Kontrolni znaki imajo tudi imena (in kratice): znak s kodo `10` se imenuje *line feed* oziroma LF.

Najbrž ste že opazili razliko med izpisom nizov s `print` in brez?

[9]:
```
ime = "Benjamin"

ime
```

[9]:
```
'Benjamin'
```

[10]:
```
print(ime)
```

```
Benjamin
```

Funkcija `print` izpiše niz. Brez `print`-a pa dobimo predstavitev niza, kakršno bi vtipkali v program, se pravi z narekovaji. Poglejte, kaj se zgodi, če poskušamo pogledati večvrstični niz `s`!

[14]:
```
s
```

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

Med `je` in `dolg` je znak s kodo 10. Ta znak nima "grafične predstavitve". Python namesto tega znaka izpiše `\n`. To sta sicer dva znaka, vendar oba skupaj pomenita samo en znak, znak s kodo 10.

Na enak način ga lahko napišemo ga lahko tudi sami.

[11]:
```
s = "Tole gre pa\nv dve vrstici"

print(s)
```

```
Tole gre pa
v dve vrstici
```

Se pravi, vedno, ko v niz napišemo `\n`, Python tega ne bo razumel kot vzvratno poševnico (*backslash*) in `n`, temveč kot znak za prehod v novo vrsto (*new line*). Izmed vseh ostalih podobnih znakov navadno uporabljamo le še enega`\t`, ki pomeni tabulator. Več o njem, ko ga bomo potrebovali.

Zaporedjem, ko sta `\n` in `\t` pravimo ubežna zaporedja (*escape sequence*). Načelno je tudi vse drugo, kar se začne z `\` ubežno zaporedje, vendar je Python dovolj pameten, da takrat, ko `\`‹ sledi kak nesmiseln znak, obdrži `\`.

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.

[12]:
```
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.

[13]:
```
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*): pred narekovaj damo črko `r` in vzvratna poševnica bo samo vzvratna poševnica.

[14]:
```
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.