06 Moduli

January 28, 2024

1 Moduli

Povedali smo že: Python ima tisoče in tisoče funkcij, zato morajo biti urejene, postavljene tja, kjer jih bomo lahko našli.

Enega od principov smo že spoznali: metode. Metode so pripete na reči (bolj učeno in pravilno: objekte) in počnejo stvari, tipične za te reči. Nizi imajo metode, kot so `lower` in `split`, slovarji pa `get` in `setdefault`.

Poleg metod ima Python tudi funkcije, ki ne spadajo k nobenemu specifičnemu tipu. Primera sta `input` in `print`. A teh je malo; privilegij ležati kar tako, na prostem, ima le nekaj funkcij.

Večina funkcij je pospravljeni v škatlice. Škatlicam se v Pythonu reče *modul* (angl. *module*). Najbrž boste slišali tudi izraz paket (*package*) in celo knjižnica (*library*). Zaradi njih se ne vznemirjajte: paket je hierarhična zbirka modulov, knjižnica pa neka zaokrožena celota različnih … modulov, paketov … česarkoli. Za nas so tule pomembni samo moduli. Izvedeli bomo, kako jih uporabljati in videli nekaj uporabnih reči v nekaj uporabnih modulih.

1.1 Uvažanje modulov

Da uporabimo funkcije v modulu, moramo modul najprej uvoziti.

Doslej smo uporabljali, na primer, funkcije iz modula `math` in uvažali smo jih s `from math import *`. Pri poučevanju programiranja se izogibam temu, da bi morali tipkati kakšne fraze, ki jih ne razumete. `from math import *` je bila ena redkih in čas je, da izvemo, za kaj gre. Ter tudi čas, da se te fraze odvadimo in se naučimo pisati kot se spodobi.

Modul z matematičnimim funkcijami, `math`, pravilno uvozimo tako:

[1]:
```
import math
```

S tem v bistvu dobimo novo spremenljivko, `math`.

[2]:
```
math
```

[2]:
```
<module 'math' from '/Users/janez/opt/miniconda3/envs/prog/lib/python3.11/lib-
dynload/math.cpython-311-darwin.so'>
```

Tole je spremenljivka najbolj čudnega tipa doslej. Ni preprosto število (`int`, `float`) ali niz (`str`) ali seznam ali terka (`list`, `tuple`), temveč spremenljivka vrste “modul” (“module”).

Funkcije pripadajo modulu, tako kot metode objektu. če imamo niz `ime`, bomo do njegove metode `upper` prišli s piko, `ime.upper`. Tu pa je podobno: če imamo modul `math`, bomo do njegove funkcije `sqrt` prišli z `math.sqrt`, do njegove spremenljivke `pi` pa z `math.pi`.

[3]:
```
math.pi
```

[3]:
```
3.141592653589793
```

[4]:
```
math.sin(math.pi / 4)
```

[4]:
```
0.7071067811865475
```

Uvozimo še en modul, `os`.

[5]:
```
import os
```

V `os` so funkcije, s katerimi lahko ustvarimo direktorij ali pobrišemo datoteko. Poleg tega pa vsebuje še nekaj zanimivega: modul. Modul `path` je modul znotraj modula `os`. Vsebuje, recimo, funkcijo `splitext`, ki za podano ime datoteke vrne osnovo in končnico.

[6]:
```
osnova, koncnica = os.path.splitext("nek_film.avi")
```

[7]:
```
osnova
```

[7]:
```
'nek_film'
```

[8]:
```
koncnica
```

[8]:
```
'.avi'
```

Tako se uvaža module.

1.1.1 Uvažanje posamičnih funkcij

Če funkcije določenega modula kličemo velikokrat in predvsem, če to počnemo v aritmetičnih izrazih, to postane nepregledno.

```
x = 2 * math.sin(math.radians(phi) + math.pi / 2) - 2 * math.cos(math.radians(alpha))
```

Boljše bi bilo imeti te funkcije pri roki brez ponavljajočega se `math`. Uvozimo jih z

[25]:
```
from math import sin, cos, radians, pi
```

[26]:
```
cos(pi / 4)
```

[26]:
```
0.7071067811865476
```

`from math import sin, cos, radians, pi` uvozi naštete funkcije in postanejo, pod temi imeni, dostopne programu. Imeli bomo torej imena `sin`, `cos`, `radians` in `pi`, ne pa tudi `tan` in `log`. Predvsem pa ne `math`. Se pravi:

- če uvozimo `import math`, imamo `math` in, recimo `math.cos`, ne pa `cos`. Uvozili smo pač `math`, ne `cos`.
- če uvozimo `from math import cos` pa imamo `cos` in ne `math` ali `math.cos`. Uvozili smo pač `cos` in ne `math`.

1.1.2 Uvažanje vsega

Obstaja še "preprostejši" način uvažanja.

[27]:
```
from math import *
```

Ta je podoben prejšnjemu, le da funkcij ne naštevamo. `*` predstavlja *vse*, kar je v modulu.

Prvi način, uvažanje modula, ima to prednost, da je v vsakem klicu funkcije jasno, odkod ta funkcija prihaja. Slabost so precej daljši izrazi.

Drugi način, uvažanje posamičnih funkcij skrajša izraze. Odkod prihaja kakšna funkcija, še vedno vidimo, vendar je potrebno za to pogledati na začetek programa, v `import`-e.

Tretji način je v splošnem slab. Izrazi so krajši, vendar za posamične funkcije ne vemo, odkod so prišle. Na ta način uvažamo kvečjemu modul `math` in nobenega drugega. Posebej v večjih projektih.

Sprememba: gornji odstavek sem napisal pred desetimi leti. V resnici že dolgo, dolgo nisem uvozil modula na takšnen način, razen v prvih tednih Programiranja 1. Tudi vam ga ni treba. To se ne dela.

1.2 Kje uvažamo

Vedno na začetku programa. Ne kar tako, vmes.

Izjemoma: na začetku funkcije. To storimo, recimo, kadar ni jasno, ali bo modul možno uvoziti oziroma je uvažanje počasno, potrebujemo pa ga le v neki funkciji. Druga situacija, kjer bi to prišlo prav, je, če se dva modula uvažata vzajemno, eden drugega. To razrešimo tako, da določen modul uvozimo šele, ko ga potrebujemo.

To ni zakon, samo dogovor.

Uvažanje modulov se v resnici zgodi samo enkrat. Če petkrat napišemo `import math`, se bo modul v resnici uvozil le prvič.

1.3 Modul `math`

Ogledali si bomo nekaj uporabnih modulov. Python jih ima približno dvesto; še tisoče in tisoče jih najdete na internetu.

Prvi je `math`, vendar o njem ne bomo povedali veliko. Ima pač vse funkcije, ki si jih ima katerikoli kalkulator … in še malo več. :) Na primer največji skupni delitelj, fakulteto in binomske koeficiente …

Priložnost izkoristimo le za to, da omenimo dve posebnosti v zvezi s števili. Podatkovni tip `float` premore poleg normalnih števil še dve posebni: neskončno (in minus neskončno) ter ni-število (*not a number*).

[13]:
```
math.inf
```

[13]:
```
inf
```

[14]:
```
-math.inf
```

[14]:
```
-inf
```

[15]:
```
math.nan
```

[15]:
```
nan
```

Prvi dve sta lahko uporabni za kako iskanje minimuma ali maksimuma.

[20]:
```
def min(s):
    m = math.inf
    for x in s:
        if x < m:
            m = x
    return m
```

`math.inf` je večji od vseh števil (razen od sebe) in torej uporabna začetna vrednost pri iskanju minimuma.

`nan` je zanimivejša reč. V nekaterih jezikih ga dobimo kot rezultat napačnih argumentov za matematične funkcije, recimo, če poskušamo izračunati koren ali logaritem negativnega števila. Pythonove funkcije v tem primeru vrnejo napako, pač pa ga vrnejo nekatere funkcije v knjižnicah (modulih), ki si jih bomo namestili dodatno, zato je prav, da veste zanj.

`nan`, *not a number*, je slepa ulica števil. Karkoli počnemo z njim - ga seštevamo, odštevamo, množimo, celo množimo z nič - vedno ostane `nan`. Še huje: `nan` ni ne večji ne manjši od nobenega števila. Niti od neskončno ni manjši (in seveda ne večji, pa tudi enak ne).

[22]:
```
math.nan < math.inf
```

[22]:
```
False
```

[23]:
```
math.nan > math.inf
```

[23]:
```
False
```

[24]:
```
math.nan == math.inf
```

[24]:
```
False
```

In, najhujše, enak ni niti sebi.

[25]:
```
math.nan == math.nan
```

[25]:
```
False
```

Kako potem preverimo, če ima neka spremenljivka vrednost **nan** - če je ne moremo primerjati niti z nan?

[32]:
```
x = math.nan

if x == math.nan:
    print("Ojoj!")
else:
    print("Vse je OK.")
```

```
Vse je OK.
```

Pač, izkoristimo njegovo neumnost proti njemu samemu.

[33]:
```
x = math.nan

if x != x:
    print("Svet se podira!")
```

```
Svet se podira!
```

Lepši, pravilnejši način, da preverimo, ali ima neka spremenljivka vrednost **nan**, je funkcija `math.isnan`.

[34]:
```
if math.isnan(x):
    print("Ojoj!")
```

```
Ojoj!
```

Na vse to se bo koristno spomniti, ko bomo delali z, na primer, kakšnimi statističnimi knjižnicami, ki bodo takrat, ko česa ne bo možno izračunati, vrnili **nan**.

Mimogrede, tole čudno obnašanje, po katerem **nan** ni enak niti samemu sebi, ni kakšna Pythonova muha, temveč del standarda IEEE 754, ki določa zapis števil s plavajočo vejico (float) in njegovo vedenje. Enako se vede vsak vzgojen jezik.

1.4 Modul `random`

Modul `random` vsebujejo funkcije, ki počnejo (psevdo-)naključne stvari. (To je: videti so naključne, vendar niso, saj je vse, kar naračunajo običajni računlniki, izračunano in ne izžrebano.) Omenili jih bomo le nekaj.

`random.random()` vrne naključno število med 0 in 1.

[1]:
```
import random

random.random()
```

[1]:
```
0.6494175036385138
```

[2]:
```
random.random()
```

[2]:
```
0.7117311287907189
```

`random.uniform(a, b)` vrne naključno število med `a` in `b`. Funkcija se imenuje `uniform`, ker gre za enakomerno porazdelitev - vsa števila so enako verjetna.

[3]:
```
random.uniform(10, 20)
```

[3]:
```
19.118192660781837
```

[4]:
```
random.uniform(10, 20)
```

[4]:
```
17.985517474304324
```

Če potrebujemo celo naključno število, pokličemo `randint`.

[9]:
```
random.randint(10, 20)
```

[9]:
```
15
```

Če nas namesto enakomerne zanima kakšna druga porazdelitev: modul `random` jih pozna veliko: beta, gama, eksponentna … in seveda tudi Gaussova.

[12]:
```
iq = random.gauss(100, 10)

iq
```

[12]:
```
98.30080171519482
```

Sestavimo seznam imen.

[13]:
```
imena = ["Ana", "Berta", "Cilka", "Dani", "Ema", "Fanči"]
```

`random.choice` vrne naključni element podanega seznama.

[14]:
```
random.choice(imena)
```

[14]:
```
'Cilka'
```

`random.sample` izbere naključni vzorec podane velikosti.

[15]:
```
random.sample(imena, 3)
```

[15]:
```
['Fanči', 'Ema', 'Ana']
```

`random.choices` je podoben, vendar se izbrani primeri lahko tudi ponavljajo. Velikost vzorca mora biti podana kot argument z imenom `k`.

[17]:
```
random.choices(imena, k=5)
```

[17]:
```
['Berta', 'Berta', 'Ana', 'Dani', 'Cilka']
```

`random.shuffle` premeša podani seznam.

[18]:
```
imena
```

[18]:
```
['Ana', 'Berta', 'Cilka', 'Dani', 'Ema', 'Fanči']
```

[19]:
```
random.shuffle(imena)

imena
```

[19]:
```
['Ana', 'Dani', 'Cilka', 'Ema', 'Fanči', 'Berta']
```

1.5 Modul os

Modul `os` vsebuje kop reči, povezanih z operacijskim sistemom. Ker o operacijskih sistemih ne vemo dovolj, večine funkcij ne bi razumeli. (Ta prva oseba množine velja tudi zame. Sicer večino razumem, o mnogih pa bi moral prebrati malo več, da bi vedel, za kaj pravzaprav gre.) Prgišče pa jih bomo vseeno redno uporabljali.

- `getcwd()` vrne trenutni direktorij. Torej direktorij, v katerem bi funkcija `open` iskala datoteko, če bi ji podali le ime datoteke.
- `chdir(path)` spremeni trenutni direktorij. Pot je lahko absolutna (s `/` na začetku) ali relativna.
- `mkdir(path)` naredi nov direktorij.
- `remove(filename)` pobriše datoteko s podanim imenom. Brez milosti. Nobenega "ali res želite pobrisati".
- `rename(name, newname)` preimenuje datoteko.
- `listdir(path)` vrne seznam vseh imen datotek v podanem direktoriju.

Največkrat boste potrebovali slednjo. Podatke iz vremenskih postaj, recimo, sem dobil v obliki tisočev datotek, ki sem jih potem prebral in (seveda filtrirane) podatke zapisal v novo datoteko.

Tole je trenutna vsebina trenutnega direktorija (v katerem so tile zapiski).

[22]:
```
import os
```

[23]:
```
os.listdir()
```

[23]:
```
['progkog 10-24.ipynb',
 'Untitled1.ipynb',
 'kolesa.txt',
 '.DS_Store',
 'kolesa2.txt',
 '03c Slovarji.ipynb',
 'temperature.txt',
 '04 seznami.ipynb',
 'vreme',
```

```
 '03a Kako racunalnik shrani besedilo.ipynb',
 '05 Moduli.ipynb',
 '02b logični izrazi.ipynb',
 '.ipynb_checkpoints',
 '02 datoteke, zanke, pogoji.ipynb',
 'december.txt',
 '03b Metode nizov.ipynb']
```

1.6 Modul `os.path`

`path` je modul znotraj modula `os`. Tudi ta ima marsikaj, nekaj funkcij pa je primernih tudi za nas.

- `os.path.exists(name)` vrne `True`, če (v trenutnem direktoriju) obstaja datoteka ali direktorij s podanim imenom.
- `os.path.isdir(name)` vrne `True`, če je podano ime direktorij (znotraj trenutnega direktorija).
- `os.path.isfile(name)` vrne `True`, če je podano ime datoteka (znotraj trenutnega direktorija).
- `os.path.splitext(name)` vrne osnovo in končnico podanega imena datoteke.

Ostale so očitne, za nas je posebej zanimiva zadnja.

Izpišimo imena vseh datotek s končnico .txt, ki se nahajajo v trenutnem direktoriju.

[24]:
```
for ime in os.listdir():
    osnova, koncnica = os.path.splitext(ime)
    if os.path.isfile(ime) and koncnica == ".txt":
        print(ime)
```

```
kolesa.txt
kolesa2.txt
temperature.txt
december.txt
```

1.7 Modul `collections`

Med vsemi zanimivimi rečmi v `collections` vsaj za zdaj omenimo le dve.

1.7.1 `defaultdict`

`defaultdict` je slovar, ki sproti dodaja neobstoječe ključe, podati pa mu moramo funkcijo, s katero si bo izmišljal njihove vrednosti. Primerne so le funkcije, ki ne sprejemajo argumentov, oziroma, točneje, funkcije, ki jih lahko pokličemo (tudi) brez argumentov. Najpogosteje bo to `int`. Če jo pokličemo brez argumentov namreč vrne `0`.

[25]:
```
int()
```

[25]:
```
0
```

[27]:
```
from collections import defaultdict
```

```
d = defaultdict(int)

d["Ana"] = 5
d["Berta"] = 3

d
```

[27]:
```
defaultdict(int, {'Ana': 5, 'Berta': 3})
```

Doslej nič posebnega, le pri izpisu nam ne pokaže samo slovarja, temveč doda še, da je to `defaultdict` s funkcijo `int`. Zabava se začne, ko povprašamo po ključu, ki ne obstaja. Tu pokliče podano funkcijo (`int`) in vrne njeno vrednost, novi par pa doda tudi v slovar.

[29]:
```
d["Cilka"]
```

[29]:
```
0
```

[30]:
```
d
```

[30]:
```
defaultdict(int, {'Ana': 5, 'Berta': 3, 'Cilka': 0})
```

Smemo celo, recimo, povečati vrednost neobstoječemu ključu.

[31]:
```
d["Dani"] += 1

d
```

[31]:
```
defaultdict(int, {'Ana': 5, 'Berta': 3, 'Cilka': 0, 'Dani': 1})
```

Preštejmo, kolikokrat se je avtor datoteke kolesa.txt vozil s katerim kolesom in koliko je prevozil z njim. Spomnimo se: vrstice datoteke predstavljajo posamične poti in vsebujejo ime kolesa, razdaljo v kilometrih in še nekaj, kar nas tu ne zanima (višino).

[32]:
```
uporaba = defaultdict(int)
pot = defaultdict(int)

for vrstica in open("kolesa.txt"):
    kolo, razdalja, _ = vrstica.split(",")
    uporaba[kolo] += 1
    pot[kolo] += int(razdalja)

print(uporaba)
print(pot)
```

```
defaultdict(<class 'int'>, {'Nakamura': 22, 'Cube': 43, 'Canyon': 26, 'Stevens':
9})
defaultdict(<class 'int'>, {'Nakamura': 439, 'Cube': 3174, 'Canyon': 2766,
'Stevens': 607})
```

Enako bi lahko naredili tudi z običajnimi slovarji, vendar bi zahtevalo nekaj `if`-ov ali pa metod, kot sta `setdefault` ali `get`.

Zdaj pa se posvetimo dražbi: radi bi sestavili slovar, katerega ključi bodo predmeti, vrednosti seznami ponudb za ta predmet. Spet bomo uporabili `defaultdict`, vendar vrednosti ne bodo `int`-i, temveč seznami. Funkcija `list` nam, če jo pokličemo brez argumentov, prikladno vrne prazen seznam.

[40]:
```
d = defaultdict(list)

d["foo"]
```

[40]:
```
[]
```

V tak seznam lahko celo dodajamo z `append`!

[41]:
```
d["bax"].append(12)
d["bax"].append(5)
```

[42]:
```
d
```

[42]:
```
defaultdict(list, {'foo': [], 'bax': [12, 5]})
```

Zdaj pa zares.

[44]:
```
ponudbe = defaultdict(list)

for vrstica in open("../domace-naloge/03-drazba-brez-anonimnosti/zapisnik.txt"):
    predmet, oseba, cena = vrstica.split(",")
    ponudbe[predmet].append(int(cena))

for predmet, cene in ponudbe.items():
    print(predmet, cene)
```

```
slika [31, 33, 35, 37, 40, 45]
pozlačen dežnik [29]
Meldrumove vaze [44, 46, 48, 53, 57, 60, 61, 63, 67, 71, 76, 78]
skodelice [50, 55, 60, 61, 62, 65, 68, 70, 74, 76, 80, 83]
kip [30, 32, 37, 39, 43, 44, 45, 50, 53, 55, 58, 61, 63, 68, 72, 76, 77, 81, 85,
86, 90, 92, 94, 97, 98, 99, 100, 103, 107]
čajnik [15]
srebrn jedilni servis [27, 30, 35, 39, 40, 45, 47, 49, 53, 55, 58, 59, 62, 63]
perzijska preproga [16, 21]
```

Kako lažje bi bile domače naloge, če bi že pred par tedni vedeli za `defaultdict`! Vendar se potem nikoli ne bi naučili zares delati z datotekami, zankami, seznami… ;)

1.7.2 `Counter`

`Counter` iz nekega razloga pišemo z veliko. (Razlog ni posebej dober. Iz istega razloga bi lahko z veliko pisali tudi `defaultdict`.

`Counter` je v sorodu z `defaultdict`-om in ga včasih nadomešča. Trenutno nimamo pri roki dobrega primera za njegovo uporabo, oziroma, točneje, da bi ga učinkovito uporabljali, bi morali znati nekaj, česar še ne znamo. Vseeno pa lahko pokažemo, kaj počne.

Recimo, da imamo seznam imen oseb, ki jim je nekdo telefoniral.

[45]:
```
klici = ["Ana", "Ana", "Berta", "Cilka", "Ana", "Cilka", "Dani", "Ema"]
```

Seveda želimo prešteti, kolikokrat je poklical koga. To bi znali kar trivialno narediti z `defaultdict`-om, s `Counter` pa je še trivialneje:

[46]:
```
from collections import Counter

stevci = Counter(klici)

stevci
```

[46]:
```
Counter({'Ana': 3, 'Cilka': 2, 'Berta': 1, 'Dani': 1, 'Ema': 1})
```

Pogosto nas bo zanimalo, koga je klical največkrat - ali pa katere tri - in za to obstaja metoda.

[47]:
```
stevci.most_common(3)
```

[47]:
```
[('Ana', 3), ('Cilka', 2), ('Berta', 1)]
```

Tole je seznam, urejen po pogostosti.

1.8 Modul `csv`

Tale vam bo všeč. Datoteke, ki vsebujejo podatke, ločene z vejicami, tako kot jih je naš `zapisnik.txt`, so kar pogoste. Tej obliki zapisa rečemo *comma separated values* ali, s kratico, csv. V to obliko zna shranjevati tudi Excel - ob opozorilu, da se bo ob tem izgubilo vse oblikovanje in še kaj. (Enkrat sem obljubil, da se bomo učili brati Excelove datoteke. To še ni to. Brali bomo tudi .xlsx. Ampak še ne.) V tej obliki je bil tudi naš zapisnik dražbe in vse druge datoteke.

Python ima zato modul csv, ki zna brati take reči.

1.8.1 `reader`

Nas zanimajo imena vseh udeležencev dražbe?

[6]:
```
import csv

udelezenci = set()
for predmet, oseba, cena in csv.reader(open("../domace-naloge/04-analiza-drazbe/
  ↪zapisnik.txt")):
```

```
    udelezenci.add(oseba)

udelezenci
```

[6]:
```
{'Ana', 'Berta', 'Cilka', 'Dani', 'Ema', 'Fanči', 'Greta', 'Helga'}
```

Funkciji `csv.reader` podamo datoteko (ne le imena, potrebno je poklicati tudi `open`). Vrne nekaj, čez kar lahko gremo z zanko `for`, pa dobivamo podatke iz vrstic. Nobenega `split`.

`csv.reader` privzame, da so podatki ločeni z vejico. Če so s čim drugim, mu to povemo z dodatnim argumentom `delimiter`. Ločilo v datoteki "kolesa.txt" iz pete domače naloge je bil `-`.

[55]:
```
ime_dat = "../domace-naloge/05-druzabno-omrezje-drazbe/kolesa.txt"

for v in csv.reader(open(ime_dat), delimiter="-"):
    print(v)
```

```
['Cube', '5031', '159', 'Janez', '2017']
['Stevens', '3819', '1284', 'Ana', '2012']
['Focus', '3823', '1921', 'Benjamin', '2019']
```

Poleg ločila ima datoteka lahko še kup drugih lastnosti. Recimo, da na dražbi prodajajo komplet *žlica, nož in vilice Ludvika XIV*. V datoteki bi tako dobili vrstico

```
žlica, nož in vilice Ludvika XIV,Ana,12945
```

in `s.split(",")` bi vrnil štiri stvari, namesto treh, pač zaradi vejice med žlico in nožem. Excel, recimo, bi v tem primeru napisal nekaj v slogu

```
"žlica, nož in vilice Ludvika XIV",Ana,12945
```

S tem, ko bi zaprl prvo polje v narekovaje, bi povedal, da gre tu za eno samo reč in da je potrebno vejico znotraj tega ignorirati. Različni programi in sistemi imajo različna pravila; temu se v jeziku funkcije `csv.reader` reče `dialect`. Privzeti dialekt je `"excel"` in ta bo za datoteke iz Excela navadno delal. Za ostale lahko uporabite `Sniffer`.

[60]:
```
sniffer = csv.Sniffer()
```

Z metodo `.read()` preberemo celotno vsebino datoteke.

[61]:
```
open(ime_dat).read()
```

[61]:
```
'Cube-5031-159-Janez-2017\nStevens-3819-1284-Ana-2012\nFocus-3823-1921-Benjamin-
2019\n'
```

`sniffer` ima metodo `sniff`, ki ji kot argument damo vsebino datoteke (ali vsaj dovolj velik kos, da je iz njega razvidno, kako je datoteka oblikovana). Sniffer bo uganil, v kakšnem slogu je napisana datoteka.

[62]:
```
dialect = sniffer.sniff(open(ime_dat).read())
```

Nato pokličemo `csv.reader` in mu poleg datoteke podamo še dialekt.

Da bo lažje vidno, naredimo vse še enkrat.

[ ]:

[65]:
```
import csv

ime_dat = "../domace-naloge/05-druzabno-omrezje-drazbe/kolesa.txt"
sniffer = csv.Sniffer()  # vrne novega "snifferja"
dialect = sniffer.sniff(open(ime_dat).read())  # preberemo vsebino datoteke in
 ↪jo damo posniffati :)
for v in csv.reader(open(ime_dat), dialect):
    print(v)
```

```
['Cube', '5031', '159', 'Janez', '2017']
['Stevens', '3819', '1284', 'Ana', '2012']
['Focus', '3823', '1921', 'Benjamin', '2019']
```

1.8.2 `DictReader`

Recimo, da bi imeli takšno datoteko s kolesi:

```
kolo,razdalja,višina,lastnik,leto nakupa
Cube,5031,159,Janez,2017
Stevens,3819,1284,Ana,2012
Focus,3823,1921,Benjamin,2019
```

Za razliko od prejšnjih datotek (in vseh, ki smo jih videli doslej) ima ta datoteka v prvi vrstici imena stolpcev. Zato jo lahko namesto z `reader` beremo z `DictReader`, ki za vsako vrstico ne vrne seznama temveč slovar, katerega ključi so imena stolpcev. To je imenitno.

[79]:
```
import csv

for vrstica in csv.DictReader(open("kolesa-z-glavo.txt")):
    print(vrstica["kolo"], ":", vrstica["leto nakupa"])
```

```
Cube : 2017
Stevens : 2012
Focus : 2019
```

Ker bodo imele vaše datoteke zelo pogosto glavo, boste pretežno uporabljali `DictReader`. To je praktično, saj je s slovarji lažje delati kot s seznami. Imena stolpcev, kot so *kolo*, *lastnik* in *leto nakupa* je lažje brati kot indekse 0, 3 in 4, pa še zmotili se ne bomo pri štetju.

1.9 Modul `statistics`

O modulu `statistics` se ne bomo preveč razgovorili. Osebno ga nikoli ne uporabljam, saj se vse to in še veliko več najde v modulih knjižnice `numpy`, s katero se da tudi veliko preprosteje narediti veliko več - a le če znaš. Python ima ta modul zgolj zato, ker je `numpy` pač ogromna dodatna knjižnica, ki je ne dobimo s Pythonom. (Po drugi strani pa si vsak resen uporabnik pythona namesti tudi `numpy`.)

Skratka: `statistics` vsebuje funkcije `mean`, `median`, `mode`, `stdev` in še kup drugih, ki znajo za podani seznam izračunati poprečje, mediano, modus, standardno deviacijo in še kup drugega.

[80]:
```
import statistics

visine = [185, 192, 160, 173, 180]
```

[81]:
```
statistics.mean(visine)
```

[81]:
```
178
```

[82]:
```
statistics.stdev(visine)
```

[82]:
```
12.227019260637483
```

Ve tudi za korelacijo in linearno regresijo, tu pa se neha. Kdor bi rad izvedel več, naj pogleda v dokumentacijo modula `statistics`.

1.10 Moduli `time`, `datetime`, `calendar`

Pri delu boste najbrž pogosto naleteli na razne datume, čase in podobno. Funkcije, povezane s tem, so razmetane po treh modulih. Vse skupaj je prilična zmešnjava. Zanjo niti ni toliko kriv Python, kot, predvsem, dejstvo, da se tu navezuje na različne standardizirane funkcije različnih sistemov. Nekatere stvari na Windows delujejo drugače kot na macOS ali na Linuxu in potem dobimo, kar imamo.

1.10.1 `time`

Modul `time` (dokumentacija) se ukvarja s trenutnim časom in drugimi časi in pretvarjanjem med njimi.

Glavna funkcija je `time`, ki vrne število sekundo, minevših od 1. januarja 1970 po Greenwichu.

[86]:
```
import time

time.time()
```

[86]:
```
1699905222.8740501
```

To je uporabno, če bi radi merili (realen) čas, ki je minil med dvema dogodkoma v programu. Vendar … no, ne zelo. Za to obstajajo boljše funkcije.

Za naše potrebe sta morda bolj uporabni `gmtime` in `localtime`.

[ ]:
```

```

[87]:
```
print(time.gmtime())
print(time.localtime())
```

```
time.struct_time(tm_year=2023, tm_mon=11, tm_mday=13, tm_hour=19, tm_min=53,
tm_sec=43, tm_wday=0, tm_yday=317, tm_isdst=0)
time.struct_time(tm_year=2023, tm_mon=11, tm_mday=13, tm_hour=20, tm_min=53,
tm_sec=43, tm_wday=0, tm_yday=317, tm_isdst=0)
```

Obe vrnete čudo istega tipa. Imenuje se `struct_time`. Tako ime kot oblika izhajata iz jezika C. Ime ni pomembno, pomembna so imena polj, ki vsebujejo trenutno leto, mesec in dan, uro, minute in sekunda, pa dan v tednu in letu, za zraven pa še polje, ki pove, ali gre za letni čas (1) ali ne (0).

[92]:
```
zdaj = time.localtime()

print("Danes je ", zdaj.tm_mday, ". ", zdaj.tm_mon, ". ", zdaj.tm_year, ".",␣
 ↪sep="")
```

```
Danes je 13. 11. 2023.
```

Modul vsebuje tudi funkcijo za oblikovanje datumov. Uporabimo jo takole:

[93]:
```
time.strftime("Danes je %d. %m. %Y, ura pa je %H.%M", zdaj)
```

[93]:
```
'Danes je 13. 11. 2023, ura pa je 20.59'
```

Podamo ji niz, ki vsebuje znake, kot so `%d`, `%m`, `%Y` in tako naprej, ter čas. Funkcija bo te znake zamenjala z ustreznimi vrednostmi.

Celoten seznam možnih znakov najdete na https://docs.python.org/3/library/time.html#time.strftime.

[95]:
```
time.strftime("Danes je %A, %d. %B %Y", zdaj)
```

[95]:
```
'Danes je Monday, 13. November 2023'
```

Tole ni izpadlo najboljše. Očitno se ni odločil za slovenščino. K temu bi se ga sicer dalo pripraviti.

[ ]:
```

```

[101]:
```
import locale

locale.setlocale(locale.LC_ALL, "sl_SI")
```

[101]:
```
'sl_SI'
```

Zdaj pa bo.

[102]:
```
time.strftime("Danes je %A, %d. %B %Y", zdaj)
```

[102]:
```
'Danes je ponedeljek, 13. november 2023'
```

Tule smo mimogrede uporabili še module `locale`, ki vsebuje vse živo povezano z različnimi navadami različnih jezikov - od imen dni in mesecev, do tega, ali uporabljajo decimalno vejico ali piko in načina zapisa valut

[104]:
```
locale.currency(42.19)
```

[104]:
```
'42,19 SIT'
```

(haha, tolarji!)

1.10.2 `datetime`

Bistvo modula `datetime` (dokumentacija) je manipulacija z datumi. Z njim lahko odštejete dva časa in izveste, koliko let, mesecev, dni, ur, minut, sekund je med njima. Ali pa pripravite objekt vrste `timedelta`, ki bo, recimo, dve meseca in tri dni, ter to prištejete k nekemu datumu.

Še pomembneje: modul zna pretvarjati čase iz nizov.

[107]:
```
from datetime import datetime

datetime.strptime("13. november 2023, 21:14", "%d. %B %Y, %H:%M")
```

[107]:
```
datetime.datetime(2023, 11, 13, 21, 14)
```

Najprej čudni import: iz modula `datetime` uvozimo "funkcijo" `datetime`. (V resnici ni funkcija, temveč tip, vendar za tole zdaj ni časa. :)

Funkcija `strptime` zahteva niz, v katerem je zapisan datum, in niz, ki pove obliko niza - spet s črkami, kot smo jih videli v `strftime`. Do delov rezultata spet dostopamo z imeni polj.

[109]:
```
cas = datetime.strptime("13. november 2023, 21:14", "%d. %B %Y, %H:%M")
```

[110]:
```
cas.day
```

[110]:
```
13
```

[111]:
```
cas.hour
```

[111]:
```
21
```

Tole vam bo znalo razkopati datum v prilično poljubni obliki. Takole opravimo z Američani.

[112]:
```
datetime.strptime("11/13/2023", "%m/%d/%Y")
```

[112]:
```
datetime.datetime(2023, 11, 13, 0, 0)
```

Takole pa z Američani, ki pišejo leto brez stoletij.

[113]:
```
datetime.strptime("11/13/23", "%m/%d/%y")
```

[113]:
```
datetime.datetime(2023, 11, 13, 0, 0)
```

1.10.3 `calendar`

Zadnji od treh modulov, povezanih s časom, je preprosti `calendar` (dokumentacije). Ta vsebuje stvari kot so imena dni, mesecev in podobno.

1.11 Modul `urllib`

(V delu. To bo še dodano. Prej ko slej. Mogoče. Najbrž.)