Python-programmi stiilijuhis (Python style guide)

Stiilijuhis *(style guide)* on soovituste ja juhtnööride kogum dokumentide vormindamiseks ja kirjutamiseks. Stiilijuhiste eesmärk on tagada omavahel seotud dokumentide ühtlus. Programmeerimiskeelte stiilijuhised annavad suuniseid programmi koodi kirjutamiseks ja vormindamiseks. Juhtnööridest kinnipidamine aitab erinevatel inimestel kirjutada programmikoodi sarnases laadis ja muuta kood üksteisele paremini loetavaks ja arusaadavaks. Üldjuhul ei ole tegu rangete eeskirjadega vaid pigem soovitustega, mis võivad aga konkreetse tarkvarafirma piires rangete reeglite staatuse omandada.

Ühe Python-programmi stiilijuhise autoriks on Pythoni autor Guido van Rossum. Sellest kirjutisest on järgnevalt tehtud lühikokkuvõte, mis meie vajadustega võiks enim ühtida. Näidetena on kasutatud samas stiilijuhises olevaid näiteid.

Tervikliku *Style Guide for Python Code* leiate järgmiselt lingilt: https://www.python.org/dev/peps/pep-0008/

Google poolt kirja pandud *Google Python Style Guide* leiad lingilt: https://google.github.io/styleguide/pyguide.html

Järgnevas kokkuvõttes on kasutatud mõtteid mõlemast juhisest. Samuti on kasutatud nende juhendite näiteid.

Python'is kirjutatud programmide stiilist - lühikokkuvõte

Järgnevalt on välja toodud väike osa koodi kirjutamise/kujundamise stiilijuhistest.

Lühikokkuvõtte eesmärgiks on:

- anda soovitusi programmi koodi "kujundamiseks";
- näidata, millele tuleb (lisaks algoritmi realiseerimisele) koodi kirjutades tähelepanu pöörata.

Miks on vaja koodikirjutamise stiilijuhiseid? Sest koodi loetakse rohkem kui teda kirjutatakse. Koodi väljanägemine peab olema järjepidev, sest see teeb koodi lugemise inimese jaoks lihtsamaks. Järjepidevus on olulisem, kui iga hinna eest stiilijuhiste järgimine. Kui täiendad kellegi teise programmikoodi, siis vaata palun eelnevalt, kuidas on kood kirjutatud ja kasuta samamoodi tühikuid, taandeid, muutujate nimesid, kommentaare jne.

Koodi paigutus

Programmi ühtset stiili aitab tagada see, kui kood on paigutatud ridadesse sarnasel viisil, on kasutatud sama suuri taandeid jne (õnneks ei pea Pythoni puhul rõhutama, et taanded üldse olulised on). Kui kahes eelnevalt märgitud allikas on soovitused väga erinevad, toon ka selle allpool välja.

Taanded

Iga taandetaseme suurus on neli (4) tühikut. Kui poolitatakse rida, et tohi taane olla sama suur, vaid soovitavalt palju suurem (et taanetega edastatavat programmi struktuuri ja ridade poolitamisi mitte segi ajada). Poolitatud rea taane võiks sellisel juhul lause loogikaga kokku sobida.

```
# Teises reas on lisatud rohkem taanet, et eristada seda
# funktsiooni sisu taandest.
def very_long_function_name(var_one, var_two,
                            var_three, var_four):
    print(var_one)
```

Tabulaator (nn Tab) ja tühik on erinevad sümbolid. Kasutatav Pythoni IDLE muudab ühe tabulaatori neljaks (4) tühikuks. Seega lõpuks on koodis siiski tühikud. Kasutades aga mõnda teist editori tee kindlaks, mis juhtub tab'ide ja tühikutega ning taga, et kasutusel oleks pidevalt sama sümbol (soovitavalt tühik).

Rea pikkus ja lausete arv reas

Koodirida peab olema inimesele mugav lugeda. Piirdu 79-80 tähemärgiga reas. Sel juhul mahub rida korraga tekstiredaktori aknasse ära ja on ka võimalik mitut akent kõrvuti lahti hoida. Samas võidakse arendusmeeskondades kokku leppida ka teistsuguste reapikkuste suhtes. Võivad olla mõned erandid – näiteks kommentaarina kirjutatud veebiaadress.

Kui rida on vaja poolitada, siis on selleks võimalik rea lõpus kasutada längkriipsu \. Google juhend siiski soovitab seda mitte kasutada ja lähtuda hoopis Pyhtoni oskusest ühendada ridu, kui reavahetus on sulu sees. See töötab nii ümar-, kandiliste kui ka looksulgude puhul. Selline näide on eelmises alapeatükis.

Ehkki Pythonis on võimalik ka mitu lauset ühele reale paigutada, on soovitav kirjutada iga lause eraldi reale. Ainsaks

aktsepteeritavaks erandiks on `if`-lauses ühe tegevuslause lisamine tingimusega samale reale. Kuid sel juhul `else`-osata:

```
if arv == 0: arv = 10
```

Käsk "import"

Käsk `import` paigutatakse alati koodi algusesse, soovitavalt iga moodul imporditakse eraldi käsuga eraldi real. Võid importida ka konkreetseid funktsioone, kuid väldi mooduli importimisel funktsioonide nimes metamärke (nt *) - sellest ei saa teada, mida imporditi. Google soovituse kohaselt tuleks mooduli importimisel pigem vältida funktsioonide ükshaaval importimist ja kasutada mooduli nime funktsiooni väljakutsumisel. See tagab paljude moodulitekasutamisel parema orienteerumise koodis.

Tühikud avaldistes ja lausetes.

Keelereeglid lubavad tühikuid üsna suvaliselt kasutada, kuid mõistlik on lähtuda mõnedest reeglitest. Kõige üldisem reegel on, et kirjavahemärkide ümber tuleks tühikuid kasutades lähtuda trükkimise reeglitest. Kuid siin on ka erandeid.

A. Tühikut ei lisata vahetult sulu sisse, koma, ette jms.

```
Sobib: spam(ham[1], {eggs: 2})
Ei sobi:  spam( ham[ 1 ], { eggs: 2 } )
Sobib: if x == 4: print(x, y); x, y = y, x
Ei sobi:  if x == 4 : print(x , y) ; x , y = y , x
```

B. Kindlasti ei lisata tühikut funktsiooni väljakutses funktsiooni nime ja sulu vahele. Samuti ei lisata tühikut kandilise sulu ette, mis sisaldab indeksit.

```
Sobib: spam(1)
Ei sobi:  spam (1)
Sobib: dict['key'] = list[index]
Ei sobi:  dict ['key'] = list [index]
```

C. Ära kirjuta üle 1 tühiku omistusmärgi ümber. Kuid soovitav on üks tühik siiski mõlemale poole omistusmärki lisada. Sobivaks ei peeta omistusmärkide joondamist, nagu järgnevas halvas näites näha on. Samuti ei soovitata joondada rea lõppudes olevaid kommentaarimärk #. Põhjuseks on suurem ebamugavus programmikoodiga töötamisel.

```
Sobib:
x = 1
y = 2
long_variable = 3
```

```
Ei sobi:
x             = 1
y             = 2
long_variable = 3
```

D. Tühikud avaldistes - lisa tühik järgmiste tehtemärkide mõlemale poole:

omistamine `(=, +=, -=` jne.`)`,

võrdlustehted `(==, <, >, !=, <>, <=, >=, in, not in, is, is not)`, loogikatehted `(and, or, not)`

Ülejäänud tehtemärkide puhul võib järgida tehete järjekorda - varem tehtavatel tehetel ei ole ümber tühikuid, hiljem tehtavatel on. Siiski selles osas on olulisem järgida üksi koodi kirjutades oma „tundeid” ja olla järjekindel.

```
Sobib:
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

```
Ei sobi:
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
```

```
c = (a + b) * (a - b)
```

E. Ei ole soovitav panna mitut lauset ühele reale, ka mitte liitlausete puhul `(if..., while ...)`

Ei ole soovitav (võib, aga ei ole ilus):

```
if foo == 'blah': do_blah_thing()
for x in lst: total += x
while t < 10: t = delay()
```

Kindlasti mitte nii:

```
if foo == 'blah': do_blah_thing()
else: do_non_blah_thing()
```

Igati sobilik:

```
if foo == 'blah':
    do_blah_thing()
else:
    do_non_blah_thing()
```

Kommentaarid

Mõned kommenteerimise põhireeglid:

- Kommentaari puudumine on parem vigasest kommentaarist.
- Kommentaarid on soovitavalt täislaused.
- Ära kommenteeri seda, mis on nagunii üheselt selge. Kommenteerides eelda, et koodi lugeja teab Pythonist ja programmeerimisest vähemalt sama palju kui sina.
- Üldiselt peavad kommentaarid olema inglisekeelsed.

NB! Kommenteerimise põhireeglite vastu on sihilikult eksitud kursuse koodinäidetes, sest näite eesmärk on kõik üksipulgi algajale lahti seletada - sellest nii eesti keele kasutus kui ka ülekommenteeritus (kommentaarid pea iga koodirea järel). Samuti võid siin kursuses ülesandeid lahendades eksida reeglite vastu, sest kirjutad neid endale ja selleks, et hiljem oma programmi lugedes kõik oluline meelde tuleks.

Nimed ehk identifikaatorid

Programmides kasutatakse mitmeid erinevaid nimekirjutamise laade. Vastavalt väga üldistele ja paljudes programmeerimikeeltes kehtivatele reeglitele võib nimi koosneda suurtest ja väikestest ladina tähtedest `(A..Z)`, `a..z)`, numbritest `(0..9)` ja allkriipsust `(_)`, sh number ei tohi olla esimene sümbol.

Sõltuvalt arvutis kehtivast keelest, kultuurist jms võib olla võimalik kasutada ka teisi tähti, nt konkreetsetes oludes saame kasutada ka nn täpilisi tähti (ö, ä, ..). **Täpitähtede kasutamisest tuleks aga hoiduda**, et mitte mõnel teisel platvormil „haiget saada“. Ka Pythoni stiilisoovitused rõhutavad eespool mainitud sümbolitega piirdumise olulisust. Samuti soovitatakse kasutada inglisekeelseid nimesid ja inglisekeelseid kommentaare. Nende soovituste vastu eksin kursusel sihilikult. Kuid kindlasti ei keela teil inglise keelt kasutada.

On mitu erinevat nime kirjutamise stiili, mida tuleb üksteisest eristada. Erinevaid stiile soovitatakse erinevat tüüpi nimede jaoks. Kommentaariks – programmis ei kasutata nimesid ainult muutujate jaoks. Näiteks on vaja nimesid ka funktsioonidele, klassidele, konstantidele jne.

- `b` (üksik väiketäht)

- `B` (üksik suurtäht)

- `lowercase` (kõik on väiketähed)

- `lower_case_with_underscores` (väiketähed koos allkriipsudega)

- `UPPERCASE` (kõik on suurtähed)

- `UPPER_CASE_WITH_UNDERSCORES` (suurtähed allkriipsudega)

- `CapitalizedWords` (suurtähtedega algavad sõnad ehk nn *CamelCase*)

NB! Kui nimes on lühend, tuleks kõik lühendi tähed kirjutada suurtena: `HTTPServerError` on parem kui

`HttpServerError`.

- `mixedCase` (võrreldes eelmisega on esitäht väike)

- `Capitalized_Words_With_Underscores` (suurtähed sõnade alguses koos allkriipsudega, peetakse inetuks)

Stiilijuhistest leiab põhjalikuma nimekirja erinevat tüüpi nimedega seotud soovitustest. Antud kursuse oleks vaja arvestada järgnevaga:

Ühetähelisi muutujaid võib kasutada vaid indeksiteks ja tsükli iteraatoriteks. Väldi ühetähelise nimega väikest `l`-i, suurt `O`-d ja suurt `I`-d, sest `l` ja `I` on visuaalselt sarnased, nagu ka `O` ja `0` (null).

Muutujate, funktsioonide ja moodulite nimedes on soovitav kasutada stiili „väiketähed koos eraldava allkriipsuga) `lower_case_with_underscores`.

Konstantide jaoks on soovituslik stiil „suurtähed koos eraldava allkriipsuga `UPPER_CASE_WITH_UNDERSCORES` (meil konstandi rollis olevad muutujad).

Kasutatud materjalid

G. van Rossum, B. Warsaw, N. Coghlan. PEP 8 -- Style Guide for Python Code. 2013
https://www.python.org/dev/peps/pep-0008/

A. Patel, A. Picard, E. Jhong, J. Hylton, M. Smart, M. Shields. Google Python Style Guide rev 2.59
https://google.github.io/styleguide/pyguide.html