Pravila kodiranja i sintaksa programskog jezika Python

Kao i kada je reč o učenju drugih programskih jezika, i Python se najefikasnije uči kroz praktične vežbe, pa će paralelno čitanje i prekucavanje linija koda iz nastavnih jedinica u radno okruženje dovesti do boljeg razumevanja samog koda i njegovog rezultata.

Jedna od značajnijih odlika Pythona je pre svega mogućnost vrlo jednostavnog čitanja koda. Pogotovo što važi pravilo gde se sam kod više puta čita nego što se piše. Pravila kodiranja i sintakse su definisana u zvaničnom PEP 8 uputstvu (Python Enhancement Proposals).

Modul this

Kako sve funkcije i promenljive nestaju prilikom završetka Python programa, može se javiti i želja da nam te definicije ostanu i za naredno pokretanje – možemo ih smestiti u poseban fajl, a takav fajl, u zavisnosti od namene, možemo nazvati i modul. Drugim rečima, modul je fajl koji sadrži upravo promenljive i funkcije koje se mogu iznova koristiti pri svakom novom pokretanju Pythona.

Ako se u komandnom okruženju Pythona (shell) otkuca komanda *import this* (naredbom *import* smo „uvezli" modul *this* u naš program), dobiće se sledeći ispis:

Primer korišćenja this naredbe
Kod
`import this`

Ovo se zove Zen Pythona ili Pythonov Zen, odnosno set **osnovnih** pravila formatiranja koda kojih bi programer trebalo da se pridržava. Iako je lista ovih pravila znatno duža, u lekciji će se obraditi samo najbitnija, a više informacija se može dobiti na linku koji vodi do zvanične dokumentacije na sajtu Pythona.

Napomena

Komentari se u Pythonu pišu koristeći znak # sa tekstom komentara koji sledi nakon njega. Python podržava samo jednolinijske komentare koristeći # znak. Ukoliko želimo da ispišemo komentar u nekoliko redova, koriste se trostruki navodnici:

```
"""ovo je
komentar u
više linija"""
```

Uvlačenje

Za uvlačenje se mogu koristiti tabulator ili prazna mesta, ali koja god opcija da se koristi – programer je se stalno mora pridržavati. Za Python 2.x verziju nije bilo bitno koji se metoda koristi, ali je od Python 3.x verzije poželjan metod korišćenje praznih mesta, jer tabulator ne

predstavlja isti broj karaktera na svakom sistemu (negde je to četiri mesta, negde dva, negde više). Zbog toga se za Python 3.x preporučuje četiri mesta po nivou uvlačenja. Jedan nivo uvlačenja predstavljaju sve linije koda koje su isto vertikalno poravnate. Ako se piše dugačak izraz, najbolje je držati ga vertikalno poravnatim.

Primer pravilnog uvlačenja koda:

Radno okruženje

```
def sum(nbr_a, nbr_b):

    return nbr_a + nbr_b # Text indented for four spaces

nbr_a=4
nbr_b=5
result = sum(nbr_a, nbr_b)
print(result)
```

Objašnjenje:

U ovom primeru, naredba `return nbr_a + nbr_b # Uvučeno četiri mesta` predstavlja jedan nivo uvlačenja.

Jedan izraz po liniji

Na prethodnom primeru može se primetiti i ovo pravilo, odnosno pravilo pisanja jednog izraza po liniji. Iako Python podržava pisanje više izraza po liniji koda, to je pogrešna praksa i trebalo bi je izbegavati:

Primer lošeg načina pisanja izraza

Kod

```
print("Number_a"); print("Number_b")

nbr_b=-5

if nbr_b  < 0: print("Number_b negative!")
```

Primer pravilnog načina pisanja izraza

Kod

```
print("Number_a")
print("Number_b")

nbr_b=-5

if nbr_b  < 0:
    print("Number_b is negative!")
```

Takođe je bitno napomenuti da se ovo pravilo odnosi na sve delove koda, pa tako i na uvoženje novih modula, za koje koristimo komandu import; zbog toga je nepravilno napisati `import sys, os`, dok bi pravilan način bio:

Primer

```
import sys
import os
```

Preporučuje se korišćenje import direktive uvek na početku fajla nakon informacija o autorskim pravima i dokumentacionih stringova. Dobra praksa, radi kasnijeg lakšeg čitanja koda, jeste i grupisanje import direktiva u grupe, i to u module koji pripadaju standardnom setu Python biblioteka, srodnim eksternim modulima kao i specifičnom uvozu klasa na nivou same biblioteke. U kasnijim lekcijama ćemo detaljno obraditi korišćenje import naredbe i uvoženje modula i biblioteka. Ove ispise je zgodno propratiti prigodnim komentarima. Primer pravilnog uvoženja eksternih modula se može videti u tabeli ispod.

Primer pravilnog uvoženja modula u program

```
# Za korišćenje sistemski specifičnih poziva
import os

# Za korišćenje regularnih ekspresija
import re

# Iz bs4 biblioteke uvesti BeautifulSoup analizator
from bs4 import BeautifulSoup
```

Spisak svih Python modula možete pogedati ovde: [Python Module Index — Python 3.11.1 documentation]

Dužina linije koda

Trenutno pravilo je da linija koda ne sme da prevaziđe 79 karaktera, dok dokumentacioni string ne sme da pređe 72 karaktera po liniji. Dugačke linije koda se mogu prelomiti koristeći karakter `\`, kao što se može videti na sledećem primeru:

Primer prelamanja dugačke linije koda u naredni red

```
a = 1 + 2 + 3 + 4 + 5 \
+ 6 + 7 + 8 + 9 \
+10

print(a)
```

Eksplicitni i implicitni kod

Iako je Python veoma fleksibilan jezik, treba imati na umu da je dizajniran sa idejom da se lako čita. Kako bi se taj uslov zadovoljio, poželjno je uvek eksplicitno pisati kod umesto koristiti implicitne pretpostavke o nameni samog koda.

U narednom primeru funkcija `sum()` „sakriva" promenljive x i y pod imenom args. Na ovaj način, funkcija dozvoljava da se pozove i sa više promenljivih nego što je potrebno.

Primer eksplicitnog zadavanja argumenata funkciji

Radno okruženje

```
def sum(*args):
    nbr_a, nbr_b = args
    return (nbr_a + nbr_b)
print(sum(2,5))
```

Sada probajte ovu situaciju ispraviti na sledeći način i pogledajte rezultat:

Primer eksplicitnog zadavanja argumenata funkciji

```
def sum(nbr_a, nbr_b):
    return (nbr_a + nbr_b)

print(sum(2,5))
```

Objašnjenje:

Ako postavimo da argument funkcije bude *args možemo joj proslediti koliko god elemenata želimo I tada će funkcija da vrati sumu elemenata koje postavimo unutar zagrada poziva funkcije, u ovom slučaju 7. A ako fiksno zadamo 2 elemenata kao u ovom primeru I prosledimo funkciji više ili manje od 2 elementa doći će do greške jer program očekuje tačno 2 elementa. Rezultat je isti kao i u prvom slučaju.

Imenovanje

Postoji nekoliko razlličitih konvencija za imenovanje modula, klasa metoda ili funkcija i promenljivih. Ta pravila obuhvataju i kombinacije malih i velikih slova sa ili bez donje crte, velikih početnih slova i drugih karaktera. Kako postoji toliko varijacija, doslednost jednom stilu ne postoji. U sledećoj tabeli vidimo te kombinacije i stilove:

Primer	Komentar
nbra = []	# mala slova
nbr_a = []	# mala slova sa donjom crtom
NBRA = []	# velika slova
NBR_A = []	# velika slova sa donjom crtom
NbrA = []	# početna velika slova
nbrA = []	# pomešan stil

Tabela 5.1. Primeri imenovanja promenljivih

Bitno je naglasiti da su sve ovo ispravni načini i da je na programeru da izabere onaj koji mu najviše leži i da ga se pridržava i bude mu dosledan kroz pisanje čitavog programa. Po PEP 8 dokumentu, primenjuju se sledeća pravila:

- imena ne smeju sadržati karaktere van ASCII tabele;
- moduli moraju imati kratka imena sa svim malim slovima;
- klase se definišu koristeći stil početnih velikih slova;
- definisane greške i izuzeci takođe koriste stil velikih početnih slova, ali sa *Error* sufiksom;
- konstante se pišu velikim slovima.

Za više detalja pogledati priručnik PEP 8.

Ostalo je još da se napomene da je „pajtonski" način imenovanja promenljivih zapravo korišćenje malih slova sa donjom crtom između.

Napomena

ASCII je akronim od American Standard Code for Information Interchange. Kako računari jedino „razumeju" brojeve, ovaj kod nam zapravo omogućava da u numeričkoj notaciji predstavimo karaktere kao što su, na primer, a, b itd. Koji su to zapravo kodovi predstavljeni u ASCII tabeli – možete videti na linku.

Pitanje

Izabrati odgovarajuću opciju. Za Python 3.x verziju se preporučuje:

- uvlačenje koda tabulatorom
- **uvlačenje koda praznim mestom**
- da se uvlačenje koda zanemari; ono nije potrebno, jer Python prevodilac ne obraća pažnju na to

Objašnjenje:

Preporučuje se da se kod uvlači sa četiri prazna mesta po nivou uvlačenja, a ne tabulatorom, jer dužina tabulatora nije ista na svim sistemima.

Zagrade u Pythonu

Python sintaksa dozvoljava rad sa različitim tipovima zagrada, koji su pojašnjeni u tabeli 5.2.

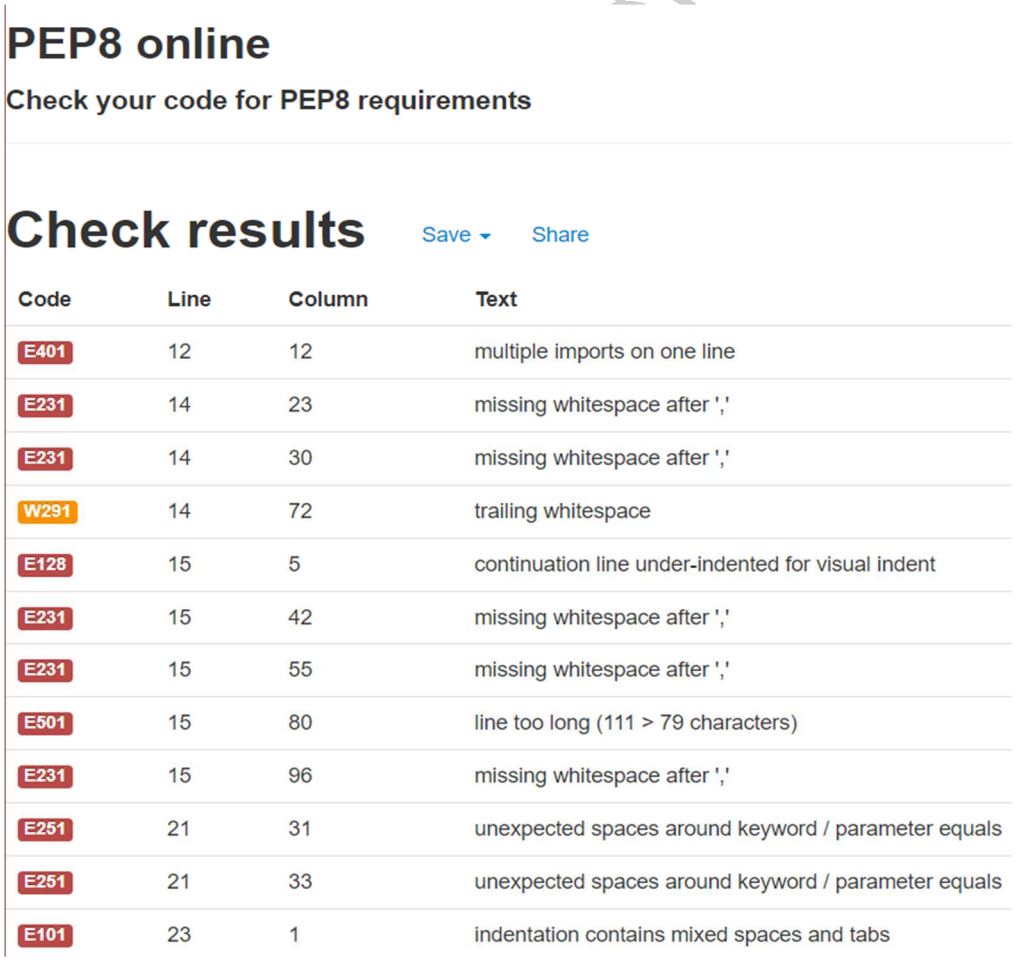
Iako će pojmovi iz ove tabele detaljno biti obrađivani u narednim nastavnim jedinicama, ova tabela može služiti kao dobra referentna tačka.

Tabela zagrada i njihovih opštih namena	
Tip zagrade	**Generalna upotreba**
()	ove zagrade koristimo prilikom poziva funkcija i u radu sa n-torkama i setovima
[]	ove zagrade koristimo u radu sa listama
{}	ove zagrade koristimo u radu sa rečnicima i setovima

Tabela 5.2. Primeri imenovanja promenljivih

Validacija stila pisanja

Smernice su odličan način za standardizaciju pisanja koda i programer bi trebalo da ih se pridržava koliko god je moguće. Postoje i rešenja koja automatski skeniraju i traže greške u stilu pisanja. Za Python 3 postoje zvanični, ali i eksterni moduli koji ovo proveravaju i to konkretno `python3-pycodestyle`, koji se može instalirati pomoću komande `pip3 install python3-pycodestyle` koja se otkucava u terminalu operativnog sistema. Lakši način za takvu proveru je korišćenje onlajn besplatnog alata po imenu PEP8 online check. Nakon unošenja našeg koda možemo dobiti povratnu informaciju o njegovoj ispravnosti, nalik na izveštaj na slici 5.1.



PEP8 online

Check your code for PEP8 requirements

Check results Save Share

Code	Line	Column	Text
E401	12	12	multiple imports on one line
E231	14	23	missing whitespace after ','
E231	14	30	missing whitespace after ','
W291	14	72	trailing whitespace
E128	15	5	continuation line under-indented for visual indent
E231	15	42	missing whitespace after ','
E231	15	55	missing whitespace after ','
E501	15	80	line too long (111 > 79 characters)
E231	15	96	missing whitespace after ','
E251	21	31	unexpected spaces around keyword / parameter equals
E251	21	33	unexpected spaces around keyword / parameter equals
E101	23	1	indentation contains mixed spaces and tabs

Slika 5.1. Rezultati analize stila koda

Zanimljivost

Često ćemo u dokumentaciji sresti termin *pythonic*. Ovo se zapravo odnosi na način pisanja koda.

Korišćenje svih mogućnosti i olakšica koje Python sintaksa nudi se naziva *pajtonski* način. Obično *nepajtonski* stil pisanja koda praktikuju programeri koji poznaju druge programske jezike a tek su počeli sa upoznavanjem Python sintakse. Ovo se najlakše vidi na primeru iteracije i množenja elemenata liste.

Nepajtonski pristup (unpythonic):

```
x=[1, 2, 3, 4, 5]
result = []
for idx in range(len(x)):
    result.append(x[idx] * 2)
print(result)
```

Pandan ovome, ali koristeći pajtonski pristup (pythonic):

```
x=[1, 2, 3, 4, 5]
result=[(element * 2) for element in x]
print(result)
```

Probajte oba načina:

Radno okruženje

Rezultat je lista koja sadrži 5 elemenata: [2, 4, 6, 8, 10], jer u novu listu ubacujemo svaki element koji kvadriramo iz stare liste.

Iako su oba primera sintaksički sasvim tačna i daju isti rezultat, samo je drugi primer pajtonski. U njemu smo umesto klasične for petlje koristili skraćeno generisanje listi (list comprehension), koje predstavlja jaču stranu Pythona.

Rezime

- Pravila kodiranja i sintakse su definisana u zvaničnom PEP 8 uputstvu.
- Neka od bitnih pravila pisanja koda u Pythonu su:
  - uvlačenje koda – poželjno je uvek koristiti četiri prazna mesta umesto samog tabulatora;
  - jedan izraz po liniji – iako Python sintaksa omogućava više izraza po liniji, poželjno je pisati jedan izraz po liniji;
  - dužina linija koda – radi lakše preglednosti i čitljivosti, dužina linije koda ne sme da prevazilazi 79 karaktera.

- Koja god konvencija imenovanja da se koristi, potrebno je ispratiti sledeća pravila:
  - imena ne smeju sadržati karaktere van ASCII tabele;
  - moduli moraju imati kratka imena sa svim malim slovima;
  - klase se definišu koristeći stil velikih početnih slova;

    - definisane greške i izuzeci takođe koriste stil velikih početnih slova, ali sa Error sufiksom;
    - konstante se pišu velikim slovima.

- Pajtonski (pythonic) način pisanja koda podrazumeva korišćenje svih mogućnosti i olakšica koje Python sintaksa nudi.