VEŽBA 7

Pregled

Moduli i paketi predstavljaju sredstvo da se na lak i pregledan način organizuje ranije napisani programski kod i konačno ponovo upotrebi u novim programima. Python pruža punu podršku za ovakav sistem rada.

Funkcije koje vraćaju funkcije – funkcije fabrike

Funkcija fabrika (*factory*) je funkcija koja vraća funkciju kao svoju povratnu vrednost. Vraćena funkcija može unutar sebe da zapamti i neke lokalne promenljive funkcije koja je okružuje (fabrika) i da ih koristi u svom radu – ovakva funkcija naziva se omotačem (*closure*). Ovu osobinu često koriste funkcije fabrike.

Moduli

Svaki program napisan u jeziku Python može koristiti programske module u svom radu. Pre nego što takav program pozove neki deo određenog modula potrebno je da izvrši uvoz (*import*) tog modula. Moduli se pišu u jeziku Python isto kao i drugi programi i obično sadrže funkcije i klase koji će u daljem tekstu biti nazivani elementima. Fajlovi koji sadrže modul treba da imaju ime koje se poklapa sa imenom modula i ekstenziju .py kao i obični programi. Na disku treba da se nalaze u istom direktorijumu kao i Python program koji ih uvozi ili u nekom direktorijumu koji je uključen u putanju pretraživanja modula.

U narednim primerima se pretpostavlja da na odgovarajućem mestu na disku postoji fajl `m_primer.py`. Iz modula se elementi mogu uvesti u različitim varijantama. Prva je da se uveze sve, a to se može zapisati na različite načine:

```
import m_primer          # nacin 1
import m_primer as imd   # nacin 2
```

Svi navedeni načini uvoze sve elemente modula `m_primer`. Pretpostavimo da modul sadrži funkcije `primer_f` koju treba pozvati nakon uvoza. U zavisnosti od načina uvoza, poziv može da se razlikuje:

```
m_primer.primer_f()      # uvoz na nacin 1
imd.primer_f()           # uvoz na nacin 2
```

Jasno je da pri korišćenju (pozivu) elementa osim njegovog imena treba navesti i ime modula, a imena se odvajaju tačkom. Uvoz korišćenjem načina 2 dozvoljava da se modulu da alternativno ime koje će se koristiti samo u okviru programskog modula u koji je uvoz izvršen.

Elementi se iz modula mogu uvesti i tako da uvezena imena postanu deo prostora imena programa u koji se uvozi. To se postiže tako što naredba uvoza počinje ključnom reči `from`:

```
from m_primer import *   # uvoz
primer_f()               # koriscenje uvezenog imena
```

Pošto su imena iz uvezenog modula postala deo tekućeg prostora imena, ime modula se ne navodi posebno. Ovo se na prvi pogled može učiniti kao pogodnost, ali treba imati u vidu da opsežno korišćenje ove mougćnosti otvara problem tzv. zagađenja prostora imena (engl. *namespace pollution*). To znači da je u nekom trenutku u nekom modulu programa aktivan ogroman broj imena (promenljivih, funkcija, klasa itd.) što povećava šanse da se neka imena greškom dupliraju. Da bi se taj problem umanjio, mogu se uvesti i samo pojedini elementi iz modula, a mogu im se dodeljivati i imena po izboru:

```
from m_primer import x_fun           # slucaj 1
from m_primer import x_fun as y_fun  # slucaj 2
```

U oba slučaja uvozi se funkcija x_fun. U drugom slučaju se uvezenom elementu dodeljuje novo ime koje važi samo u programskom modulu u koji je uvoz izvršen. Funkcija se poziva ovako:

```
x_fun()     # slucaj 1
y_fun()     # slucaj 2
```

Bez korišćenja forme koja počinje sa `from` nije moguće uvesti pojedinačne elemente iz modula. Drugim rečima, ovakva forma služi za uvoženje izabranih elemenata u tekući prostor imena. Forma koja počinje sa `import` zapravo uvozi ceo modul i pripadnost nekog elementa modulu treba da bude eksplicitno naznačena pri svakoj upotrebi tog elementa.

Paketi

Paketi uvode dodatni nivo organizacije u module tako što se moduli mogu grupisati u pakete, a paketi u dodatne pakete. Paketi se organizuju u strukturu stabla na isti način kao i fajlovi u direktorijume. Zapravo ime paketa biće određeno imenom direktorijuma u kojem se njegovi moduli nalaze. Ime paketa se od imena modula odvaja tačkom.

U narednim primerima se pretpostavlja da na putanji `pakA/pakB/` postoji fajl `modC.py`.

```
import pakA.pakB.modC                      # nacin 1
from pakA.pakB.modC import superX          # nacin 2
from pakA.pakB.modC import superX as sX    # nacin 3
```

Funkcija superX se nakon ovoga u zavisnosti od izabranog načina uvoza može pozvati kao:

```
pakA.pakB.modC.superX()  # nacin 1
superX()                 # nacin 2
sX()                     # nacin 3
```

Pri pisanju modula koji će biti grupisani u pakete neophodno je uraditi još jednu stvar. U direktorijumu koji sadrži module i na taj način predstavlja paket obavezno treba da se nalazi fajl pod imenom `__init__.py`. Ovaj fajl može biti i prazan. Ako nije prazan, može da obezbedi dodatnu funkcionalnost pri uvozu modula. Njegovo prisustvo je obavezno i označava da direktorijum sadrži Python module.

Obrada parametara sa komandne linije

Svi parametri zadati na komandnoj liniji naći će se u listi pod imenom `argv`. Ova lista se generiše automatski nakon pokretanja Python programa, međutim, za pristup ovoj listi neophodno je u program uključiti ugrađeni modul `sys`. Npr:

```
import sys
# ...
print(sys.argv[0])  # ispis 0-tog parametra (ime programa)
print(sys.argv[1])  # ispis prvog parametra
```

Rad sa fajlovima

Python poseduje napredne mogućnosti rada sa fajlovima. Primenjuje se objektni pristup. Funkcije koje slede ugrađene su u Python i za njihovo korišćenje nije potreban uvoz nijednog dodatnog modula.

***Funkcija* `open`**

Pre bilo kakve operacije sa fajlovima potrebno je fajl otvoriti, a to se postiže kreiranjem odgovarajućeg fajl objekta. Uobičajeni način kreiranja fajl objekta je poziv funkcije open koja nakon uspešnog otvaranja fajla takav objekat vraća. Način pozivanja je sledeći:

```
fob = open(<string_ime_fajla>, <string_pristup>)
```

Ukoliko dođe do greške pri otvaranju fajla, desiće se greška klase `IOError`. String string_pristup se sastoji iz karaktera iz sledećeg skupa: {`r` – čitanje, `w` – upis, `a` – proširivanje, `x` – fajl će biti uspešno otvoren za upis samo ako fajl pod tim imenom još ne postoji, `t` – tekst mod, `b` – binarni mod}.

***Objekat* `file`**

Ovaj objekat vraća prethodno opisana funkcija open. Sam objekat se ponaša kao generatorska funkcija koja vraća po jednu liniju tekstualnog fajla ako se upotrebi unutar for petlje.

```
for lin in fob:
  print lin
```

Ovo parče programa ispisaće sadržaj tekstualnog fajla na terminal red po red. Objekat `fob` treba da je prethodno kreiran otvaranjem fajla pomoću funkcije `open`. Sledi opis metoda ovog objekta pomoću kojih se manipuliše fajlovima.

***Metoda* `read`**

Čitanje karaktera ili bajta iz fajla.

Presretanje grešaka u Python-u

U programiranju se javljaju različiti tipovi grešaka. Sintaksne greške predstavljaju situaciju u kojoj prevodilac ili interpreter za određeni jezik ne mogu da protumače određene delove teksta programa jer nisu u skladu sa određenim pravilima tog jezika. Ovakve greške je neophodno otkloniti pre nego što će program moći da bude preveden ili izvršen i taj posao je u izvesnoj meri olakšan porukama prevodioca ili interpretera (daje se razlog greške i linija u kojoj se nalazi).

Drugi tip greške posledica je fizičke nemogućnosti da se neka operacija obavi. Na primer: promenljiva kojoj se pristupa nije definisana, pristupa se elementu liste koji je van opsega liste, pokušava se deljenje sa nulom ili se zahteva otvaranje fajla koji ne postoji. Ovakve greške se ne mogu utvrditi ili predvideti pre pokretanja programa jer one nastaju u toku rada programa – engl. *Runtime error*. Onog trenutka kada se dese, interpreter ili operativni sistem daju poruku o grešci

koja više ili manje tačno ukazuje na njen uzrok. Da bi se izbeglo njihovo pojavljivanje u toku rada programa postoje dve mogućnosti:

i. Ova mogućnost je nezavisna od programskog jezika i podrazumeva pisanje delova programa koji proveravaju postoji li mogućnost da se kritični deo programa uspešno završi. Na primer, u skladu sa prethodnim primerom: pre pristupa promenljivoj proverava se njena definisanost, pre pristupa elementu liste proverava se dužina liste, pre deljenja proveri se da li je delitelj različit od nule i pre otvaranja falja proverava se njegovo postojanje. Ova rešenja su korektna, međutim ona čine program teže čitljivim jer se u njega unose elementi koji se ne tiču algoritma rada, nego služe isključivo za sprečavanje pojave grešaka u toku rada.

ii. U nekim programskim jezicima (Python spada u takve jezike) postoji mogućnost „*hvatanja*“ grešaka. Nakon što se greška desi, ona može biti uhvaćena, nakon čega će se izvršiti deo programa koji će adekvatno reagovati na tu grešku. Taj deo takođe piše autor programa. U nastavku sledi opis hvatanja grešaka u Python-u.

```
try:
  blok programa u kom se greske hvataju
except klasa_greske_1:
  blok za reagovanje na greške tipa klasa_greske_1
except klasa_greske_2:
  blok za reagovanje na greške tipa klasa_greske_2
except:
  blok za reagovanje na sve ostale greške
```

Blok programa u kojem će greške, ukoliko se dese, biti uhvaćene ograničen je na blok iza ključne reči `try`. Svaka greška koja se može desiti spada u određenu klasu. Na primer, greška deljenja sa nulom pripada klasi `ZeroDivisionError`. Ako je pokušano otvaranje fajla koji ne postoji greška će pripadati klasi `FileNotFoundError`. Obrada, odnosno reakcija na grešku sledi u jednom od blokova navedenih iza ključne reči `except`. Ako se desi greška određene klase, izvršavanje programa će se nastaviti od odgovarajućeg except bloka. Poslednji (ili jedini) blok može biti naveden bez klase greške i on će biti izvršen ako greška ne spada ni u jednu klasu definisanu prethodnim except blokovima.

Ovakvi blokovi se mogu naći u svim delovima programa, što uključuje i funkcije svih nivoa, kao i module i metode objekata. Blokovi mogu biti i ugnežđeni, tj. jedan blok može sadržati druge blokove. Ako neki blok ne hvata grešku određenog tipa, ona se prosleđuje bloku koji taj blok okružuje itd. Najširi (podrazumevani) blok je ceo program, a greške koje do tog nivoa stignu zaustavljaju program i objavljuju se kao greške.

Zadaci

1. Napisati program koji čita sadržaj nekog tekstualnog fajla karakter-po-karakter, a pročitani karakter ispisuje na terminal. Ako je karakter malo slovo, onda ga treba pre ispisa konvertovati u veliko slovo. Ime fajla se zadaje kao argument iz komandne linije. Pripremiti fajl za testiranje (napisati proizvoljan tekst u 4 reda) i testirati program. Predvideti odgovarajuće presretanje greške ako fajl pod navedenim imenom ne postoji.

2. Napisati funkcije za izračunavanje faktorijela celog broja – $n!$ kao i izračunavanje celobrojnog stepena proizvoljnog broja (celog ili u pokretnom zarezu) – x^n . Koristeći te

dve funkcije kao i formule za razvoj u Maclaurinov red date u dodatku napisati funkcije za izračunavanje trigonometrijskih funkcija sin i cos u domenu $[-\pi,\pi]$. Ove četiri funkcije imenaovati redom: `fktrl`, `intpwr`, `mlsin`, `mlcos`. U realnoj situaciji broj sabiraka u Maclorinovom redu ne može biti beskonačan. Njihov broj (N) zadavati kao drugi argument funkcija `mlsin` i `mlcos`.

Za testiranje uraditi sledeće: generisati brojeve u intervalu od 0 do 1,5 sa razmakom 0,05 i za svaku od tih vrednosti ispisati rezultat funkcija `mlsin` i `mlcos` za broj iteracija N = 5. Uporediti dobijene rezultate sa rezultatima funkcija `sin` i `cos` iz modula `math`. Sve četiri vrednosti za jednu istu vrednost ugla ispisivati u jednom redu na širini od 10 karaktera sa 6 decimalnih mesta.

3. Funkcije iz prethodna dva zadatka uključiti u modul `funk` sadržan u paketu `vezba7`. Nakon toga napisati program koji ih importuje i testira na isti način kao i u zadatku 1.

4. Napisati program koji otvara za čitanje tekstualni fajl input.dat koji u svakom redu sadrži po jedan broj. Svaki od tih brojeva interpretirati kao broj u pokretnom zarezu – za konverziju stringa u broj u pokretnom zarezu koristiti funkciju `float()`. Korišćenjem funkcija iz modula vezba7.funk (prethodni zadatak) izračunati i ispisati vrednost sin i cos za učitani broj ako je on u domenu $[-\pi,\pi]$, u suprotnom dati odgovarajuće obaveštenje.

   Pošto funkcija `float()` izaziva izuzetak ukoliko string ne predstavlja validan broj, predvideti obradu izuzetka pomoću `try – catch` strukture. Kao i u slučaju argumenta izvan domena i za neispravnu brojnu vrednost dati odgovarajuće obaveštenje na izlazu.

Dodaci

Izračunavanje funkcije sin preko Maclaurinovog reda

$$\sin x=\sum_{i=0}^{\infty}\frac{(-1)^i x^{2i+1}}{(2i+1)!}=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots$$

Izračunavanje funkcije cos preko Maclaurinovog reda

$$\cos x=\sum_{i=0}^{\infty}\frac{(-1)^i x^{2i}}{(2i)!}=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots$$

Faktorijel celog broja

$$m!=1\cdot 2\cdot 3\cdot 4\cdot\ldots\cdot(m-1)\cdot m \quad ,$$

pri čemu je po definiciji $0!=1$.

Algoritam izračunavanja (Maclaurinovog) reda

```
function mlsin(x, n)
{
  r = 0
  i = 0
  while (i<n)
  {
```

```
      r = r + (-1^i)*(x^(2*i+1))/(2*i+1)!
      i = i + 1
    }
    return r
  }
```

Primedba: operacija stepenovanja je u pseudokodu označena sa ^ predstavlja stepenovanje, a faktorijel je označen sa !. U Python programu ih treba zameniti odgovarajućim napisanim funkcijama.