Skriptni jezici

Prof. dr. sc. Zoran Kalafatić

Doc. dr. sc. Siniša Šegvić

Zoran.Kalafatic@fer.hr, Sinisa.Segvic@fer.hr

ZEMRIS – šk.g. 2007/08

Skriptni jezici

- Sadržaj predmeta
  - ljuska operacijskog sustava i programiranje skripti
  - uvod u programski jezik Perl
  - uvod u programski jezik Python

Motivacija i ciljevi

- u svakodnevnoj programerskoj/inženjerskoj praksi pojavljuje se potreba za automatiziranjem nekih uobičajenih poslova – može značajno poboljšati produktivnost
  - jednostavne skripte u ljusci operacijskog sustava
  - upoznati se s osnovnim naredbama
  - upoznati se s osnovnim konceptima pisanja skripti
  - znati pročitati i prilagoditi postojeće skripte
- čest zadatak s kojim se susreću znanstvenici/inženjeri/programeri je obrada velikih količina podataka, često u tekstnom obliku
  - često se radi o pronalaženju zadanih uzoraka teksta, te njihovoj analizi ili pretvaranju u drugi oblik – prikladan za daljnju obradu
  - skriptni jezici, u kombinaciji s odgovarajućim alatima za obradu teksta, pokazuju se vrlo prikladnim izborom za ovakve zadatke
  - upoznati se s osnovnim konceptima i alatima za obradu teksta u skriptnim jezicima (npr. primjena regularnih izraza)

Motivacija i ciljevi (2)

- u području automatizirane obrade teksta, *Perl* je vrlo raširen alat (*Practical Extraction and Report Language*)
- u doba Interneta skriptni jezici su posebno dobili na značenju zbog podrške oblikovanju dinamičkih web stranica (generiranje *HTML*-a, mogućnost pozivanja skripti putem *CGI* sučelja)
  - *Perl* vrlo raširen
  - upoznati se s osnovama programskog jezika *Perl*
  - znati napisati jednostavnije skripte
  - moći pročitati (jednostavniji) tuđi kôd

Motivacija i ciljevi (3)

- *Python* je jedan od najpopularnijih programskih jezika današnjice
  - za razliku od Perla – vrlo jednostavne i jasne sintakse
  - sve češće se preporučuje kao prvi programski jezik
  - zamjena za *pseudo-kôd*
  - prikladan i za “ne-profesionalne” programere
  - izuzetno bogate biblioteke najrazličitijih alata daju mu ogromnu snagu
  - ima vrlo široko područje primjene – od jednostavnih skripti (zamjena za ljusku OS-a ili Perl), do oblikovanja vrlo složenih programskih sustava(podrška OOP)
- predmet nije zamišljen kao napredni tečaj, već kao upoznavanje s osnovnim konceptima koje će studenti koji budu trebali takvu vještinu lako usavršiti

Literatura

- mnoštvo literature na Internetu
- slajdovi će biti raspoloživi na stranici predmeta
- “službeno” navedena literatura:
  1. Hans Petter Langtangen,
     Python Scripting for Computational Science, Springer, 2004
  2. Stephen Kochan, Patrick Wood,
     Unix Shell Programming, 3rd edition, Sams, 2003
  3. James Lee, Simon Cozens, Peter Wainwright,
     Beginning Perl, 2nd edition, Apress, 2004
     (http://www.perl.org/books/beginning-perl/)

Organizacija nastave

- predavanja – 2 sata tjedno
- laboratorijske vježbe – 3 bloka
  - samostalan rad, predaja u za to dogovorenim terminima
- domaće zadaće
  - podloga za samostalan rad
  - priprema za laboratorijske vježbe
- provjere znanja
  - 2 međuispita i završni ispit
  - laboratorijske vježbe
  - kratke provjere znanja

Ocjenjivanje

- međuispiti:
  - 1. MI – 20 bodova
  - 2. MI – 25 bodova
- laboratorijske vježbe i domaće zadaće:
  - 20 bodova
- aktivno sudjelovanje na predavanjima:
  - 5 bodova
- završni ispit
  - uvjet za izlazak na završni ispit: do tada prikupljenih 25 bodova
  - ZI – 30 bodova
- uvjet za polaganje predmeta:
  - ukupno ostvarenih barem 50 bodova
  - barem 25% bodova ostvarenih na završnom ispitu (7.5 bodova)

Ocjenjivanje (2)

- ponovljeni završni ispit
  - mogućnost popravljanja (ali i spuštanja) ocjene
- formiranje ocjena nakon ponovljenog završnog ispita
  - 15% ocjena 5
  - 35% ocjena 4
  - 35% ocjena 3
  - 15% ocjena 2

Što je to uopće skripta ?

- program vrlo visoke razine, najčešće vrlo kratak
- napisan u skriptnom jeziku visoke razine
- primjeri skriptnih jezika:
  - Unix ljuske (*shell*), Tcl, Perl, Python, Ruby, Scheme, Rexx, JavaScript, VisualBasic,...
- mi ćemo upoznati:
  - Unix ljusku
  - Perl
  - Python

Primjeri skripti

- pamćenje parametara za poziv neke naredbe

```
#!/bin/sh
# $Id: ps2pdf14 2236 2002-02-21 21:49:28Z giles $
# Convert PostScript to PDF 1.4 (Acrobat 5-and-later compatible).

exec ps2pdfwr -dCompatibilityLevel=1.4 "$@"
```

- pozivanje niza naredbi

```
latex $1
dvips -Ppdf -o $1.ps $1
ps2pdf14 -sPAPERSIZE=a4 $1.ps $1.pdf
```

```
$ napravi skriptni1
```

Primjeri primjene

- jedan tipičan primjer – automatizacija simulacije i vizualizacije
  - priprema (generiranje) ulaznih parametara za simulacijski program
  - pokretati simulaciju, pohranjivati (organizirati) izlazne datoteke (simulacija može jako dugo trajati !)
  - priprema izlaznih podataka u oblik prikladan za program za vizualizaciju
  - pokretati program za vizualizaciju, pohraniti dobivene grafove
  - moguće uključiti i neki oblik analize podataka – primjerice na temelju rezultata simulacije odabrati posebno zanimljive rezultate

Osobine skriptnih jezika

- tipično se koristi za povezivanje više samostalnih programa u cjelinu (*glue language*)
- često se koriste za intenzivnu obradu (manipulaciju) teksta
- omogućuju manipulaciju datotečnim sustavom
- često su koriste za pisanje programa usmjerenih specifičnim potrebama korisnika
- obično se ne radi o složenim programima – relativno jednostavan razvoj
- često podržavaju oblikovanje prilagođenog grafičkog sučelja
- portabilnost (Unix, Windows, Mac)
- program se *interpretira*

Zašto skriptni jezici

- zašto ne ostati pri *uobičajenim* programskim jezicima (Java, C/C++) ?
- prednosti skriptnih jezika:
  - kraći programi
  - brži razvoj programa
  - nema deklaracija varijabli (ali zato puno provjera tijekom izvođenja programa !!!)
  - mnoštvo alata za povezivanje programa ("lijepljenje")
  - mnoštvo alata za obradu teksta
  - jednostavna izgradnja korisničkog sučelja (GUI)
  - alati za Web programiranje
- popularnost skriptnih jezika brzo raste

Zašto skriptni jezici (2)

- to ne znači da se u potpunosti trebamo odreći *sustavskih* programskih jezika
- svaka kategorija jezika ima svoje područje primjene
- skriptni jezici ciljaju na čim bržu i jednostavniju izgradnju programa
  - koriste se bogate biblioteke gotovih *komponenti* koje se povezuju u cjelinu
  - *komponentno* programiranje
- sustavski jezici nastoje osigurati čim veću učinkovitost konačnog kôda
  - mogućnost upravljanje svim detaljima izvedbe
  - prikladni za izradu komponenti koje će se povezati u skriptnom jeziku, pogotovo za računski zahtjevne dijelove funkcionalnosti

Primjer

- napisati program koji učitava niz slika i prati objekte u tom slijedu slika
- program za analizu slika napisat ćemo u C-u, a slike se mogu čitati sa standardnog ulaza (`stdin`) – na koji se može preusmjeriti sadržaj niza slikovnih datoteka ili pak slike dobivene s digitalizatora slike
- rezultat (slika s označenim objektom) može se zapisati na standardni izlaz (`stdout`) koji se može preusmjeriti u datoteku ili u program za prikaz slike
- kratka skripta koja pokreće te komponente:
  `cat slika*.ras | prati_objekt | display`
  poziva se s: `pokreni_pracenje`

Primjer (2)

- ako napišemo i program (u C-u) koji pristupa digitalizatoru slike, preuzima sliku po sliku, te ih u odgovarajućem formatu prosljeđuje na `stdout`, njegov izlaz možemo ulančati s ulazom programa za praćenje objekata:
  `grab | prati_objekt | display`
- možemo napisati program (u C-u) za predobradu ulazne slike (npr. glađenje Gaussovim filtrom zadanog parametra σ) koja sa `stdin` čita sliku, a glađenu sliku zapisuje na `stdout`:
  `grab | gladixy -s 1.5 | prati_objekt | display`
- ako se pojavi potreba za demonstracijom programa, lako je pripremiti skriptu koja učitava nekoliko karakterističnih sekvenci iz datoteka, i pokreće program za praćenje s odgovarajućim parametrima, te ispisuje na ekranu odgovarajuće poruke
- možemo dodati i grafičko korisničko sučelje (npr. složeno u Pythonu)

Kratki povijesni pregled

- 70-tih i 80-tih: Unix ljuske, jezici za upravljanje zadacima (*job control languages*)
- 90-tih: razvoj jezika *Perl*, *Python*, *Ruby*, *Tcl*
- Y2K: skriptni jezici stabilni i moćni na većini najrašireniji platformi
- u tekućem desetljeću – daljni porast interesa
- skriptni jezici podržavaju višestruku iskoristivost programa (*software reuse*)
- pojednostavljuje se izgradnja grafičkih korisničkih sučelja
- na današnjim računalima u većini primjena skriptni jezici zadovoljavaju brzinom
- Python podržava ne samo pisanje kratkih skripti već i gradnju složenijih sustava

Skripte rezultiraju kratkim kôdom

- primjer: učitati niz realnih brojeva iz datoteke, pri čemu svaki redak može sadržavati niz brojeva proizvoljne duljine

```
1.1   9  5.2
1.12398E-02
0 0.01 0.001

937
```

- rješenje u Pythonu:

```
F = open(filename, 'r')
n = F.read().split()
```

Skripte rezultiraju kratkim kôdom

- rješenje u Perlu:

```
open F, $filename;
$s = join "", <F>;
@n = split /\s+/, $s;
```

- u C++ ili Javi trebamo barem petlju, dok C ili Fortran zahtijevaju znatno više linija koda

Primjena regularnih izraza

- Perl i Python podržavaju lako baratanje regularnim izrazima – vrlo korisno za obradu tekstnih podataka
- primjer: želimo iz niza znakova pročitati kompleksne brojeve zapisane u tekstnom obliku (“dva broja odvojena zarezom i okružena zagradama, pri čemu se u broju može naći sve osim zareza, prije i poslije broja može biti proizvoljan broj praznina”)

```
(-3, 1.4) ili (-1.4376E-9, 7.11) ili ( 4, 2)
```

- rješenje u Pythonu:

```
import re
m = re.search(r'\(\s*([^,]+)\s*,\s*([^,]+)\s*\)',
              '(-3,1.4)')
re, im = [float(x) for x in m.groups()]
```

- rješenje u Perlu:

```
$s="(-3, 1.4)";
($re,$im)= $s=~ /\(\s*([^,]+)\s*,\s*([^,]+)\s*\)/;
```

Primjena regularnih izraza

- regularni izrazi (Uvod u teoriju računarstva !) su moćan jezik za specificiranje uzoraka teksta:

```
\(\s*([^,]+)\s*,\s*([^,]+)\s*\)
```

- bez podrške regularnih izraza (npr. Fortran ili C) potrebna je znatna količina programiranja kako bi se riješio isti problem
- još jedan primjer primjene regularnih izraza – kratka skripta za preimenovanje datoteka s nastavkom `.jpg` u datoteke s nastavkom `.jpeg`

```
for file in *.jpg; do
mv $file $(echo $file|sed 's/.jpg$/.jpeg/');
done
```

Nema deklaracije varijabli

- primjer funkcije u Pythonu:

```
def debug(leading_text, variable):
    if os.environ.get('MYDEBUG', '0') == '1':
        print leading_text, variable
```

- funkcija ispisuje navedenu varijablu (broj, listu, strukturu)
- ispis se može uključiti odnosno isključiti postavljanjem varijable okoline (*environment variable*) `MYDEBUG`

Ista funkcija u C++

- možemo oponašati dinamički tipiziran jezik primjenom predložaka (*templates*)
- ipak rješenje nije tako elegantno

```
template <class T>
void debug(std::ostream& o,
           const std::string& leading_text,
           const T& variable)
{
  char* c = getenv("MYDEBUG");
  bool defined = false;
  if (c != NULL) { // if MYDEBUG is defined ...
    if (std::string(c) == "1") { // if MYDEBUG is true ...
      defined = true;
    }
  }
  if (defined) {
    o << leading_text << " " << variable << std::endl;
  }
}
```

Primjena OOP za parametrizaciju tipova

- parametrizacija tipova može se postići i primjenom objektno-orijentiranog programiranja
- uvodi se osnovna klasa `A` i niz podklasa s (virtualnom) funkcijom za ispis vrijednosti
- funkciju `debug` pozivamo s argumentom `var` koja je referenca na osnovnu klasu `A`
- na ovaj način `debug` se može pozivati za sve podklase od `A`
- prednost: potpuna kontrola tipova varijabli koje `debug` smije ispisivati (može biti značajno u velikim sustavima kako bi se osiguralo da funkcija obavlja transakcije samo s određenim tipovima objekata)
- nedostatak: puno više posla, više koda, manja mogućnost uporabe napisane funkcije u novim situacijama

Brži razvoj programa

- pisanje skripti daje kraći kôd
- kraći kôd daje manju mogućnost pogreške (pokazuje se kroz iskustvo)
- brži ciklus provjere: uređivanje koda i testiranje (bez prevođenja i povezivanja)

Fleksibilna sučelja funkcija

- “prijateljske” okoline (npr. Matlab, Maple, Mathematica, S-Plus, ...) nude fleksibilna sučelja funkcija
- elementarna primjena:

```
# f is some data
plot(f)
```

- dodatno podešavanje iscrtavanja:

```
plot(f, label='f', xrange=[0,10])
```

- daljnje (finije) podešavanje:

```
plot(f, label='f', xrange=[0,10], title='f demo',
     linetype='dashed', linecolor='red')
```

Imenovani argumenti

- Imenovani argumenti = argumenti funkcije koji se zadaju ključnim riječima i pretpostavljenim (*default*) vrijednostima, npr.

```
def plot(data, label='', xrange=None, title='',
         linetype='solid', linecolor='black', ...)
```

- redoslijed i broj argumenata u pozivu funkcije može odabrati korisnik

Ispitivanje tipa varijable

- unutar funkcije može se ispitati tip varijable koju je korisnik proslijedio
- `xrange` može se izostaviti (vrijednost `None`), ili može biti zadan u obliku liste s 2 elementa (xmin/xmax), zadan nizom 'xmin:xmax', ili zadan kao jedan broj (što označava raspon 0:broj) itd.

```
def plot(data, label='', xrange=None):
  if xrange is not None: # i.e. xrange is specified by the user
    if isinstance(xrange, list): # list [xmin,xmax] ?
      xmin = xrange[0]; xmax = xrange[1]
    elif isinstance(xrange, str): # string 'xmin:xmax' ?
      xmin, xmax = re.search(r'(.*):(.*)',xrange).groups()
    elif isinstance(xrange, float): # just a float?
      xmin = 0; xmax = xrange
```

Klasifikacija programskih jezika (1)

- za razredbu programskih jezika mogu se primijeniti različiti kriteriji
- jezici s dinamičkim odnosno statičkim dodjeljivanjem tipova (*dynamically vs statically typed languages*)
- Python (dinamički):

```
c = 1           # c je integer
c = [1,2,3]     # c je lista
```

- C (statički):

```
double c; c = 5.2;  /* c prima samo double */
c = "a string..."   /* compiler error */
```

Klasifikacija programskih jezika (2)

- slabo odnosno jako (strogo) tipizirani jezici
  (*weakly vs strongly typed languages*)
- Perl (slabo):

```
$b = '1.2'
$c = 5*$b;     # implicitna pretvorba tipova: '1.2' -> 1.2
```

- Python (jako):

```
b = '1.2'
c = 5*b        # nedozvoljeno; nema implicitne pretvorbe tipova
               # zapravo rezultat je ponavljanje niza
               # '1.21.21.21.21.2'
```

Klasifikacija programskih jezika (3)

- interpretirani odnosno prevođeni jezici
- dinamički odnosno statički tipizirani jezici – može li varijabla mijenjati tip ?
- jezici visoke odnosno niske razine (Python-C)
- jezici vrlo visoke razine nasuprot jezika visoke razine (Python-C++)
- skriptni nasuprot sustavskih jezika
  (scripting vs system languages)

Konstrukcija i izvođenje izvršnih datoteka (1)

- kôd se može konstruirati i pokretati tijekom izvođenja
- primjer: želimo učitati datoteku oblika:

```
a = 1.2
no of iterations = 100
solution strategy = 'implicit'
c1 = 0
c2 = 0.1
A = 4
c3 = StringFunction('A*sin(x)')
```

- na temelju učitane datoteke želimo postaviti vrijednosti varijabli (`a`, `no_of_iterations`, `solution_strategy`, `c1`, `c2`, `A`)
- možemo li izvršiti funkciju zadanu u datoteci ?
  kod skriptnih jezika možemo

Konstrukcija i izvođenje izvršnih datoteka (2)

- Rješenje se može postići kratkim (*generičkim*) komadićem kôda:

```
file = open('inputfile.dat', 'r')
for line in file:
   # first replace blanks on the left-hand side of = by _
   variable, value = [word.strip() for word in line.split('=')]
   variable = variable.replace(' ', '_')
   pycode = variable + '=' + value
   exec pycode  # obavimo naredbu konstruiranu iz teksta
```

- ovo ne možemo napraviti u jezicima kao što su Fortran, C, C++ ili Java!

Lako oblikovanje grafičkog korisničkog sučelja

- Python ima sučelja prema mnoštvu biblioteka za oblikovanje krisničkih sučelja
  Gtk, Qt, MFC, java.awt, java.swing, wxWindows, Tk
- najjednostavnija biblioteka *Tk* – omogućava se izuzetno jednostavno i brzo oblikovanje grafičkog korisničkog sučelja
- brza priprema korisničkog sučelja za skripte koje pozivaju razne module, npr. priprema demonstracije

GUI: Python vs C

- Primjer: otvoriti prozor s tekstom 'Hello World'
- C + X11: 176 linija koda
- Python + Tk: 6 linija lako čitljivog koda

```
#!/usr/bin/env python
from Tkinter import *
root = Tk()
Label(root, text='Hello, World!',
foreground="white", background="black").pack()
root.mainloop()
```

- Java i C++ također zahtijevaju više koda nego Python + Tk

Primjer Tcl vs. C++ (1)

- usporedba iz poznatog članka Johna Ousterhouta (tvorca Tcl/Tk), “Scripting: Higher-Level Programming for the 21st Century” (link)
- Database application
- C++ version implemented first
- Tcl version had more functionality
- C++ version: 2 months
- Tcl version: 1 day
- Effort ratio: 60

Primjer Tcl vs. C++ (2)

- Database library
- C++ version implemented first
- C++ version: 2-3 months
- Tcl version: 1 week
- Effort ratio: 8-12