LUGT Vortragsreihe

Die Skriptsprachen Perl und Python

Hans-Jürgen Husel (Perl)
Carsten Geckeler (Python)

Tübingen, 07.12.2004

Skriptsprache Perl

Skriptsprachen

- Skriptsprachen werden interpretiert
- Schnelle Entwicklung, langsame Laufzeit
- Verbindung zwischen Applikationen und Bibliotheken

Perl

- Systemadministration, Datenbankabfragen, CGI, Netzwerk
- Entstanden aus Shell-Skript, awk, sed und C
- Perl ist für viele Plattformen verfügbar: Unix, Windows, VMS, ...
- Entwickelt von Larry Wall
- 1987 Version 1.000, 1994 Version 5.000
- Zukunft: Perl 6, Bytecode, Parrot-Interpreter (VM)

Skriptsprache Python

Python

- Guido van Rossum: BDFL = Benevolent Dictator for Life
- 1989: Beginn der Entwicklung
- 2000: Version 1.6
- Seit 2000: Version 2.0 → 3.0
- Aktuell: Python 2.4 (30. Nov 2004)
- Starker Einfluss von ABC:
  - Strukturierung durch Einrücken
  - Immutability
  - Interaktiver Modus
- Klarer, einfacher und lesbarer Syntax
- Hohe Portabilität:
  Unix/Linux, Windows, Mac, PalmOS, RiscOS, QNX, OS/2, OS/390, AS/400, PlayStation, BeOS, VMS, Sharp Zaurus, ...
- Python Software Foundation (PSF)

Datentypen / Immutability

Trennung Variable/Objekt:
Variable ist immer ein Zeiger auf ein Objekt

“Immutability”
- mutable: Objektinhalt kann verändert werden
- immutable: Objektinhalt bleibt unverändert

Typen:
- Zahlen: Int, Long, Float, Complex
- Container: Tuple, List, Dictionary
- Sonstige: File, None, regular expressions

```
tuple = (4,”df”)
list = [1,”eins”,2,”zwei”,4,”vier”]
dictionary = {1:”eins”, ”zwei”:2, (4,”df”): ”tuple”}
print list[3], dictionary[tuple]
zwei tuple
```

Datentypen (Skalare, Arrays), undef

Skalar

- Integer, Double, String, Filehandle, Referenz (Pointer)
- Skalar kann den Wert undef (nicht initialisiert) haben
- `$i = 12; $str = "Hello world";`

Array (Liste)

- geordnete Liste von Skalaren, referenziert über Index
- `@arr = (1, 3.141, "foo"); $arr[1] = 2; print $a[2];`

Hash

- ungeordnete Liste von Skalaren, Index ist String
- ```
  %h = ( "rot"  => 0xff0000,
         "grün" => 0x00ff00,
         "blau" => 0x0000ff );
  ```
- `$h{"lila"} = 0xff00ff;`
  ```

Beispielprogramm

```
#!/usr/bin/perl -w
# Dieses Programm durchsucht die Passwd Datei und
# gibt alle vorkommenden Login Shells aus
my ($sh);
my $passwd = "/etc/passwd";                # Passwd Datei
# Aufruf der Unterfunktion. Array mit Login Shells wird zurückgegeben.
my @shells = process_passwd($passwd);
# Alle Shells in einer Schleife ausgeben
foreach $sh (@shells) {
    print "Login Shell: $sh\n";
}
# Unterfunktion
sub process_passwd {
    my $file = shift(@_);                  # Übergebenes Argument
    my ($line,$sh,@shells,%tmp,@arr);      # lokale Variablen
    open(F,"<$file");                      # Datei öffnen
    while( $line = <F> ) {                 # zeilenweise lesen
        chomp($line);                      # CR/LF entfernen
        @arr = split(/:/,$line);           # String in Array splitten
        $tmp{$arr[6]} = 1;                 # Login Shell in Hash schreiben
    }
    close(F);                              # Datei schliessen
    foreach $sh (keys %tmp) {              # Schleife über alle Schlüssel des Hash
        push(@shells,$sh);                 # Schlüssel in Array schreiben
    }
    return(@shells);                       # Array mit Shells zurückgeben
}
```

Beispielprogramm

```
#!/usr/bin/python

# Dieses Programm durchsucht die Passwd Datei und
# gibt alle vorkommenden Login Shells aus

passwd = "/etc/passwd";                     # Passwd Datei

# Unterfunktion
def process_passwd(file):
    tmp = {}                                # temporäres Dictionary
    f = open(file)                          # Datei öffnen
    for line in f.readlines():              # zeilenweise lesen
        arr = line.strip().split(":")       # CR/LF entfernen, in Liste splitten
        tmp[arr[6]] = None                  # Login Shell in Dictionary schreiben
    f.close()                               # Datei schließen
    shells = []                             # leere Schell-Liste erzeugen
    for sh in tmp.keys():                   # Schleife über alle Indizes
        shells.append(sh)                   # Schlüssel an Liste anhängen
    return shells                           # Array mit Shells zurückgeben

# Aufruf der Unterfunktion. Array mit Login Shells wird zurückgegeben.
shells = process_passwd(passwd)

# Alle Shells in einer Schleife ausgeben
for sh in shells:
    print "Login Shell:", sh
```

Komplexe Datenstrukturen

Komplexe, mehrdimensionale Datenstrukturen recht einfach.

```
LoL = [ [ "fred", "barney" ],
        [ "george", "jane", "elroy" ],
        [ "homer", "marge", "bart" ] ]
DoD = { ”python”: {”author”: ”guido”}
        ”perl”: {”author”: ”larry”} }
lang = DoD[”perl”]
print LoL[2][2], DoD[”python”][”author”]
bart guido
print lang
{'author': 'larry'}
print lang[”author”]
larry
```

Komplexe Datenstrukturen

Komplexe, mehrdimensionale Datenstrukturen enthalten Referenzen auf Arrays bzw. Hashes.

```
@arr = (1,2,4,16);
$ref1 = \@arr;              # Referenz
$ref1->[4] = 32
$ref2 = [1,2,4,16,32]
$arr[4] ≡ $ref1->[4] ≡ $ref2->[4] ≡ 32

@AoA = ( [ "fred", "barney" ],
         [ "george", "jane", "elroy" ],
         [ "homer", "marge", "bart" ] );

print $AoA[2][2];
         bart
```

Bibliotheken, Objektorientierung

Definition einer Bibliothek (package) mit Funktionen

```
package Animal;
sub hi { print “hi\n” };
sub speak {
    my $class = shift;
    print "a $class goes ", $class->sound, "!\n"
}
```

Aufruf:

```
use Animal;
Animal::hi(); Animal->hi();
```

Bibliothek mit Vererbung

```
package Cow;
@ISA = qw(Animal);
sub sound { "moooo" }
```

Aufruf:

```
Cow->speak();
  a Cow goes moooo
```

Objektorientierte Programmierung

- Alle Datentypen sind Klassen

```
print "pYtHoN".lower().capitalize()
Python
```

- OO bereits seit erster Version enthalten
- Mehrfache Vererbung, Operatorüberlagerung möglich
- Vererbung auch mit Standardklassen möglich

```
class MyDict(dict):                      # Subklasse von dict
    def __setitem__(self, key, val): # self[key] = val
        dict.__setitem__(self, key.lower(), val)
    def __getitem__(self, key):          # self[key]
        return dict.__getitem__(self, key.lower())

d = MyDict()
d["Python"] = "Guido"
print d["python"], d["pYtHoN"]
Guido Guido
```

Exception Handling

```
import sys
from exceptions import *

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError, (errno, strerror):
    print "I/O error(%s): %s" % (errno, strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
```

- Exceptions sind Klassen
- Es können eigene Exceptions-Klassen definiert werden

eval, Exception Handling

Mit eval kann zur Laufzeit Code generiert und ausgeführt (interpretiert) werden.

```
$str = 'print “hello world”';
eval $str;
hello world
```

Damit ist Exception Handling möglich:

```
use English;
eval { $answer = $a / $b; };
if( $EVAL_ERROR != “” ) {
   print “Achtung: Nicht durch Null teilen!\n”;
}
```

regular expressions

Reguläre Ausdrücke sind ein wichtiger Bestandteil von Perl.
Beispiel: Text parsen

```
$str = 'blabla Time: 13:24:01 foobar'
if ($str =~ /Time: (..):(..):(..)/) {
   $hours = $1;
    $minutes = $2;
    $seconds = $3;
}
```

Beispiel: Suchen und ersetzen

```
$str = 'Das ist ein Beispiel';
$str =~ s/[aeiou]/?/g;
print “$str\n”;

D?s ?st ??n B??sp??l
```

Regular expressions

- Regular expressions werden im **`re`**-Modul bereitgestellt
- Kein spezieller Syntax für regular expressions

Beispiel: Text parsen

```
import re
text = "blabla Time: 13:24:01 foobar"
m = re.search(r"Time: (..):(..):(..)", text)
if m:
    hours, minutes, seconds = map(int, m.groups((1,2,3)))
```

Beispiel: Suchen und ersetzen

```
text = "Das ist ein Beispiel"
text = re.sub(r"[aeiou]", "?", text)
print text
D?s ?st ??n B??sp??l
```

Module, Erweiterungen

Standard-Bibliothek:

- Im Python-Kern sind nur wenig Funktionalitäten enthalten
- Zusätzliche Funktionen sind in Module ausgelagert
- Umfangreiche Standard-Bibliothek (über 200 Module)

Erweiterbar mit C-Schnittstelle:

- SWIG
- **`python setup.py install`**

Beispiel: Numerical Python

```
from Numeric import *
def dist(x,y): return (x-5)**2+(y-5)**2
m = fromfunction(dist, (1000,1000))
inv_m = 1/m              # elementweise Kehrbruch
```

Bibliotheken

- Perl lässt sich relativ leicht mit C-Bibliotheken erweitern
- Schnittstelle zwischen Perl und C setzt die Datentypen um
- Schnittstelle wird “halbautomatisch” generiert
- Bei CPAN (www.cpan.org) gibt es für alles eine Bibliothek
- CPAN Standards für Installation

```
perl Makefile.PL
make
make test
make install
```

Netzwerk-Programmierung

Socket

```
$iaddr = inet_aton(“web.de”);
$paddr = sockaddr_in(80, $iaddr);
$proto = getprotobyname('tcp');
socket(SOCK, PF_INET, SOCK_STREAM, $proto);
connect(SOCK, $paddr);
while (defined($line = <SOCK>)) {
   print $line;
}
close (SOCK);
```

Telnet

```
use Net::Telnet ();
$t = new Net::Telnet(Timeout=>10,Prompt => '/bash\$ $/');
$t->open("sparky");
$t->login($username, $passwd);
@lines = $t->cmd("/usr/bin/who");
print @lines;
$t->close;
```

Netzwerk-Programmierung

Socket:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((”web.de”, 80))
s.send(”hello\n”)
while 1:
    data = s.recv(1024)
    if not data: break
    print data
```

Telnet:

```
import telnetlib
tn = telnetlib.Telnet(”localhost”)
tn.read_until(”login: ”); tn.write(”geckeler\n”)
tn.read_until(”password: ”); tn.write(”foobar\n”)
tn.write("ls\n")
tn.write("exit\n")
print tn.read_all()
```

CGI / Mailman / Zope

CGI:

```
import cgi
form = cgi.FieldStorage()
if not (form.has_key("name") and form.has_key("addr")):
    print "<H1>Error</H1>"
    print "Please fill in the name and addr fields."
    return
print "<p>name:", form["name"].value
print "<p>addr:", form["addr"].value
```

Mailman:

- Komfortables Mailinglistenprogramm mit Web-Interface

Zope:

- Web Application Server
- dynamische Webseiten
- Vordefinierte Features: Mitgliedschaften, Suche, News

CGI

- CGI-Skripte sind wichtiger Einsatzbereich von Perl
- Erzeugung von HTML
- Parameter aus HTML-Formularen auslesen
- mod_perl für Apache: “vorkompilierte” Skripte
- Apache Module (Handler) in Perl

```
print table(Tr(td(“aaa”),td(“bbb”)));
print textfield(“vorname”);

<table><tr><td>aaa</td> <td>bbb</td></tr></table>
<input type=”text” name=”vorname” />

$vorname = param(“vorname”);
```

Datenbank-Interface

- Generisches Datenbankinterface (DBI)
- Treiber für jede Datenbank (DBD)

```
$dbh=DBI->connect("DBI:mysql:database=$db;
  host=$host",$user,$passwd);

$array_ref = $dbh->selectall_arrayref(
  "select id,name from users");

foreach $ref (@$array_ref) {
   print "ID ".$ref->[0].
         "Name ".$ref->[1]."\n";
}
$dbh->do("delete from users");
```

Datenbanken / GUIs

Datenbanken:

- MySQL, Oracle, SAP, Informix, PostgresSQL, Ingres, ...
- Python Database API Specification v2.0:
  generische API für Datenbanken

GUIs:

- Tkinter: Tk
- wxPython: wxWidgets
- PyQt: Qt
- PyGTK: GTK

```
from Tkinter import *
root = Tk()
hellolabel = Label(root, text="Hello World!")
hellolabel.pack()
root.mainloop()
```

XML

Python-Module basierend auf

- Expat
- SAX

```
import xml.parsers.expat
def start_element(name, attrs): print 'Start element:', name, attrs
def end_element(name): print 'End element:', name
def char_data(data): print 'Character data:', repr(data)

p = xml.parsers.expat.ParserCreate()
p.StartElementHandler = start_element
p.EndElementHandler = end_element
p.CharacterDataHandler = char_data

p.Parse("""<?xml version="1.0"?>
<parent id="top"><child1 name="paul">Text goes here</parent>""", 1)

Start element: parent {'id': 'top'}
Character data: 'Text goes here'
End element: parent
```

XML

Perl-Bibliotheken auf Basis von:

- Expat
- SAX
- libxml2 (Gnome)
- libxslt (Gnome)
- Xpath

```
# alle Paragraphen <p>...</p> ausgeben
my $xp = XML::XPath->new(filename=>'test.xhtml');
my $nodeset = $xp->find('/html/body/p');
foreach my $node ($nodeset->get_nodelist) {
   print "FOUND\n\n",
   XML::XPath::XMLParser::as_string($node),"\n";
}
```

Fazit

Contra

- Keine Lehr und Lernsprache
- Implizit definierte Variablen ($_, $!, ...)
- “There's more than one way to do it”
- Code kann schwer lesbar sein

Pro

- Sprache um Aufgaben zu lösen
- Einfache, effiziente, vollständige Sprache
- Reguläre Ausdrücke
- Keine Probleme mit Speicherallokation und -freigabe
- Bibliotheken für jedes Problem
- Verknüpfung von komplexen Applikationen (DB, Webserver,...)
- “Schweizer Kettensäge”

Fazit

Contra

- Strukturierung durch Einrücken der Zeilen
- (noch) nicht so viel Module wie für Perl erhältlich
- Nicht für Einzeiler (**`perl -ne '...'`**) geeignet

Pro

- Klarer, einfacher und gut lesbarer Syntax
- Gute OO-Einbindung
- Interaktiver Modus zum schnellen Testen
- “suitable as first language” / “suitable as last language :-)”
- Sehr aktive Community
- Auch für größere Projekte gut geeignet (glue language)
- Jython: Python-Implementation in 100% Java