Tcl vs. Python

Wähle das passende Werkzeug für dein Problem

Uwe Berger
Frank Hofmann

„Kampfablauf“

- Mein Weg zu Tcl bzw. Python
- „Hello World...“ in Tcl bzw. Python
- Der Kampf „Tcl vs. Python“
- Performance-Betrachtungen
- Fazit

Mein Weg zu Tcl

Uwe Berger:

- Wer bin ich...?
- Warum Tcl?
  - Kurzversion → *"Tcl und Tk. Entwicklung grafischer Benutzerschnittstellen für X Window System"; John K. Ousterhout; Addison-Wesley*
  - Langversion → *Wie viel Zeit haben wir...?*
- Wozu verwende ich Tcl hauptsächlich?

Mein Weg zu Python

Frank Hofmann:

- IT-Spezialist, langjähriger Linux-Enthusiast und Python-Trainer
- Warum Python?
  - einfach zu schreiben
  - lesbarer Code (auch noch nach längerer Zeit)
  - hohe Flexibilität (Sprachkonzepte für alle Situationen)
  - in sehr kurzer Zeit ein brauchbares Ergebnis
  - Stabilität der Sprache (wenige Änderungen)
- Wozu verwende ich Python hauptsächlich?
  Skripting und Automatisierung im Alltag

„Hello world...“ in Tcl/Tk

```
# Tcl...
puts "Hello world..."

# Tk...
label  .l -text "Hello world.."
button .b -text "Exit" -command {exit}
pack .l .b
```

- Interpretersprache(n):
  - Tcl: Tool command language → `/usr/bin/tclsh`
  - Tk: Toolkit → `/usr/bin/wish`
- John Ousterhout, ca. 1988
- Open Source; für viele Betriebssysteme verfügbar
- Aktuelle Version: 8.6.4 (03/2015)

Tcl-Grundsätze

- Es gilt immer(!): *Befehlswort Parameter1 Parameter2 …*
- *„Alles ist ein String!“*
- Reihenfolge der Kommandosubstitution kann durch eckige Klammern gesteuert [...] werden
- geschweifte Klammern {...} schützen den Inhalt vor der (ersten) Kommandosubstitution
- Variablen:
  - Keine explizite Definition erforderlich
  - $-Zeichen vor Bezeichner meint Variableninhalt
  - Nur im entsprechenden Kontextbereich gültig

Tcl-Grundsätze („Beiwerk...“)

- Befehlsende: Zeilenende oder Semikolon
- Befehl über mehrere Zeilen: Backslash \ am Zeilenende
- Kommentarzeichen (ist ein Befehl!): Rautezeichen #

- ...ansonsten gilt, wie auch bei anderen Programmiersprachen: *RTFM!*

Tcl/Tk: „coole“ Sachen

- Listen
- Einfache Client/Server-Programmierung
- Ereignisse und deren einfache Behandlung
- Dynamische Code-Erzeugung und -Abarbeitung
- Einfache Programmierung grafischer Oberflächen
- ...man findet immer wieder neue interessante Dinge
- Einfach mal lesen:
  - „How Tcl is special“: http://wiki.tcl.tk/1867
  - „Is Tcl Different“: http://wiki.tcl.tk/540
  - „Tcl/Tk is too easy“: http://wiki.tcl.tk/3222

Tcl/Tk-Informationsquellen

- "Tcl und Tk. Entwicklung grafischer Benutzerschnittstellen für X Window System"; John K. Ousterhout; Addison-Wesley
- "Effektiv Tcl/Tk programmieren"; Harrison, McLennan; Addison-Wesley
- http://www.tcl.tk
- http://wiki.tcl.tk
- Tcl-Newsgroup (engl.): comp.lang.tcl

„Hello world...“ in Python

```
print ("Hello world...")

a = b = c = d = 1
e = 17.5
f = ''Berlin''
print (('''a,e,f: %i, %f, %s''') % (a,e,f))
output = f * 4
```

- Interpretersprache:
  - Python 2.x → `/usr/bin/python`
  - Python 3.x → `/usr/bin/python3`
- Guido van Rossum, 1989/1990
- Open Source; für viele Betriebssysteme verfügbar
- Aktuelle Version: 3.5 (09/2015)

Python-Grundsätze

- Ziel: große Einfachheit, Übersichtlichkeit und Lesbarkeit
  - Relativ wenige Schlüsselwörter und Klammern
  - Zugehörigkeit (Scope) zu einem Block anhand einer einheitlichen Einrückung des Programmcodes mittels Leerzeichen oder Tabulatoren
- Variablen:
  - Keine explizite Definition erforderlich
  - Nur im entsprechenden Kontextbereich gültig
  - Dynamische Typisierung, d.h. keine besondere Kennung für Datentypen
  - Einzel- und Mehrfachzuweisung
- Umfangreiche Funktionsbibliothek

Zen of Python (Tim Peters) (Teil 1)

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

Zen of Python (Tim Peters) (Teil 2)

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than right now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

Python: „coole“ Sachen

- Listen und Listenverarbeitung für Zahlenwerte, Strings, Aufzählungen (Listen, Mengen, assoziative Arrays, Objekt-IDs)
- Umfangreiche Standardbibliothek sowie Erweiterungen
- Lesenswert:
  - The Python Practice Book: http://anandology.com/python-practice-book/index.html
  - Practical Programming: An Introduction to Computer Science Using Python 3 (Pragmatic Programmers)
  - Rance D. Necaise: Data Structures and Algorithms Using Python

Der Kampf „Tcl vs. Python“

- Runden:
  - Listenverarbeitung
  - Client/Server-Anwendungen
  - Funktionsnamen als Parameter übermitteln (Funktionszeiger)
  - Sicherheit gegen „Codeinjection“
  - Ansteuerung der serielle Schnittstelle
  - Grafische Oberflächen
  - Spracherweiterungen
- Zusatzrunden (wenn noch Zeit ist...):
  - „Zeitsteuerung“ (Tcl-Befehl *after*)
  - Arbeiten mit Dateien, Verzeichnissen
  - „Dynamischer Code“
  - Datenbankanbindung (sqlite)

Der Kampf „Tcl vs. Python“ → Hinweis

Die Quelltexte der folgenden Beispiele (und teilweise noch ein wenig mehr) ist hier zu finden:

- Uwes Tcl-Beispiele
  → https://github.com/boerge42/Tcl-Magie
- Franks Python-Beispiele:
  → https://github.com/hofmannedv/training-python

„Tcl vs. Python“ → Listen

- Beispiel Informationen zu einer Textdatei:
  - Anzahl Zeichen
  - Anzahl Zeilen
  - Anzahl Worte
  - Anzahl unterschiedliche Worte
- Sortieren von Listen

„Tcl vs. Python“ → Client/Server

- Client:
  - Kommando einlesen von stdin
  - Verbindung zum Server aufbauen
  - Kommando an Server senden
  - Ergebnis von Server empfangen und ausgeben
  - Verbindung schließen
- Server:
  - Socket definieren
  - Kommando über Socket empfangen
  - Kommando ausführen
  - Ergebnis zurücksenden

„Tcl vs. Python“ → Funktionszeiger

- Zwei Funktionen:
  - Quadrat berechnen
  - Kubikzahl berechnen
- Eine weitere Funktion:
  - Mit vier Parameter:
    - Zwei Bereichsgrenzen, einen Schrittwert
    - Namen einer Funktion, die mit einem Wert aus dem Bereich als Parameter aufgerufen werden soll
  - ruft für jeden Wert des Bereichs die benannte Funktion auf

„Tcl vs. Python“ → „Codeinjection“?

- Gemeint ist, Absicherung des Scripts vor Abarbeitung von ungewollt/boshaft eingeschleusten Schadcode
- Tcl-Stichwort: sicherer Interpreter
- Python-Stichwort: ???
- Beispiel:
  - Nur explizit gewollte Kommandos im Server (vorhergehende Runde) ausführen

„Tcl vs. Python“ → serielle Schnittstelle

- Beispiel Ansteuerung eines Arduino Nano über die serielle Schnittstelle:
  - LED an-/ausschalten:
    → Befehl: *on, off, toggle*
    → Antwort: *LED on* bzw. *LED off*
  - LED-Zustand abfragen:
    → Befehl: *status*
    → Antwort: *LED on* bzw. *LED off*

„Tcl vs. Python“ → GUIs

- Eine grafische Oberfläche (GUI) für die Bedienung des seriell erreichbaren Mikrocontrollers (MC) aus der „Vorrunde“
- Zusätzliche Funktionalität des MC:
  - Hall-IC angeschlossen; MC sendet je nach Zustand:
    → *HALL on* bzw. *HALL off*
- Funktionen GUI:
  - Eingabe/Senden Befehl an MC und Ausgabe der Antwort
  - Visualisierung des Zustandes von LED und Hall-IC

„Tcl vs. Python“ → Spracherweiterung

- Tcl:
  - Viele Erweiterungspakete verfügbar
  - Spracherweiterungen selbst schreiben:
    - in Tcl ;-)
    - in z.B. C (mittels Tcl-API)
      → „von Hand“ (Bsp. Tcl-Befehl *factorial*)
      → „Automatisiert“ z.B. mit swig (http://swig.org)
- Python:
  - Verwendung von Funktionen
  - Verwendung von Klassen und Methoden
  - swig (http://swig.org) etc.

„Tcl vs. Python“ → „Zeitsteuerung“

- Tcl: Befehl *after*
  - Starten von n Tasks, welche zyklisch durch Timer-Event getriggert werden
  - Start einer weiteren „Überwachungs-Task“, welche im Sekundentakt Informationen zu den anderen Tasks ausgibt

„Tcl vs. Python“ → Filesystem etc.

- Einfache Umsetzung des Kommandos *du -a <dir>*:
- Ermittlung von:
  → Anzahl Verzeichnisse
  → Anzahl Dateien
  → Größe aller Dateien
  ab einem, in der Kommandozeile anzugebenden, Verzeichnis

„Tcl vs. Python“ → Datenbank (sqlite)

- Öffnen/Erzeugung einer sqlite-Datei
- Erstellung einer Tabelle
- Befüllen der Tabelle mit Datensätzen
- Abfrage der Datensätze

„Tcl vs. Python“ → dynamischer Code

- Beispiel: Generierung/Verwendung von Variablennamen zur Programmlaufzeit

Performance-Betrachtungen

- Beispiel: mehrmalige Berechnung der Fakultät und Messung der Verarbeitungszeit in:
  - Tcl
  - Python
  - Tcl mit C-Extention für Befehl *factorial <number>*
  - C (zum Vergleich)

„Und der Sieger ist?...“

„...es gibt nicht den Sieger, denn...“