Programmiersprachen
Vorlesung 11 vom 18.12.23
Skriptsprachen

Christoph Lüth

Universität Bremen

Wintersemester 2023/24

Wo sind wir?

Was ist das?

- Kennzeichen von Skriptsprachen:
  1. Komposition komplexer Subsysteme zu einem Gesamtsystem;
  2. Hohe Abstraktionsstufe gegenüber der Hardware;
  3. Kleiner Sprachkern;
  4. Schneller Entwicklungszyklus;
  5. Nicht vorranging effizient, meist interpretiert;
  6. Meist auf ein Anwendungsgebiet zugeschnitten (**domänenspezifisch**).
- Kein **Programmierparadigma**, eher Benutzungspragmatik.

Zitat:

A scripting language is one where the main effect of a program is to drive another system, while in a programming language the program itself is the main action.

Quelle: `http://www.cs.man.ac.uk/~pjj/cs211/langdes/script.html`

Warum?

- Die meisten Skriptsprachen entstehen **evolutionär**:
  - Große Anwendungen mit komplexer Funktionalität
  - Anwender muss Ablauf steuern
  - Repetitiv und schematisch — Bedarf der Automatisierung
- Skriptsprachen **automatisieren**, was der Benutzer händisch macht.

Geschichtliches

- Stapelverabeitung (**batch control**) für Mainframe-Rechner: JCL (IBM) (1960?)
  "JCL ... is, I am convinced, the worst computer programming language ever devised by anybody, anywhere. It was developed under my supervision; there is blame enough to go around among all the supervisory levels." — Fred Brooks.
- Mainframes wurden zu Minicomputern zu PCs ...
  - Benutzerinteraktion durch eine **Shell**
  - Dazu Shell-Sprachen: sh, REXX (ab 1970)
- Später kamen als Anwendungsgebiete dazu: GUIs, Datenbanken, Office-Anwendungen, Webanwendungen (ab 1980)

Moderne Skriptsprachen

- **Allzweckskriptsprachen**:
  - Python, Ruby
- "**Glue languages**":
  - Tcl, bash
- Spezielle **Anwendungsgebiete**:
  - PHP, JavaScript (Webanwendungen)
  - Office-Anwendungen (Visual Basic)
  - Textverarbeitung (awk, sed, Perl)
- **Eingebettete** Sprachen:
  - Lua, Tcl, JavaScript

Schlüsselkonzepte

- Gute Unterstützung von **Zeichenketten**
  - Leichtgewichtige Parsierung durch **reguläre Ausdrücke**
  - Repräsentation strukturierter Daten durch XML oder JSON
- Integration von **Hostanwendungen**
  - GUI, Webschnittstelle, ...
  - Machen die "eigentliche" Arbeit
- Dynamisch getypt oder ungetypt
  - Flexibler, schnellerer Entwicklungszyklus

Beispielsprache: Shell

- Die **Bourne-Shell** und die **Bourne-Again-Shell**
- Erste Version 1976 für Unix V7
- Reimplementierung als Bourne-Again-Shell (`bash`) für GNU/Linux ab 1999.
- Was macht eine **Shell**?
  - Behandlung der **Benutzereingabe** (am Terminal oder auf der Kommandozeile)
  - Starten von Kommandos, Parameterexpansion, Umgebungsvariablen
    - Standardeingabe/-ausgabe
    - Als besondere "Filedeskriptoren"
  - Signalbehandlung (Unterbrechen, Suspendieren)
  - Umleitung der Ein/Ausgabe, Verkettung durch Pipes, Umgebungsvariablen
  - ... und sie kann **programmiert** werden!

Die Shell-Sprache

- Datentypen: Zeichenketten.
  - Komplett ungetypt (Was braucht man mehr?)
  - Ganze Zahlen sind auch nur Zeichenketten
  - Auswertung von Ausdrücken durch *arithmetic expansion*
- Elaborate Unterstützung von Literalen:

```
echo "Hello $World."
echo 'Hello $World.'
```

- Macros (Antiquotation)

```
x=`ls -la`
```

- Syntax: von ALGOL inspiriert

Programmierstrukturen

- Variablen (global, lokal)
- Volle Turingmächtigkeit
- `while`, `if`, `case`
- Ausführungsmodell:
  - Sequentielle Ausführung von Kommandos
  - Parameterexpansion
- Funktionen:
  - Positionale Parameterübergabe
  - Dynamisches Scoping

Beispiele:

- Aus dem echten Leben ...
- Fakultät iterativ (`fac1.sh`)
- Fakultät rekursiv (`fac2.sh`)

Beispielsprache: Tcl

Was ist Tcl?

- Tcl ist eine **Skriptsprache**
  - Dynamisch und komplett ungetypt
  - Als **eingebette** und **erweiterbare** "glue language" konzipiert
- Erfolgreich durch
  - Expect
  - Tk, das GUI Toolkit (Tcl/Tk)
- Inzwischen etwas überholt (insbesondere Tk)

Geschichte

- Entstanden 1987, Autor John Ousterhout (damals University of California at Berkeley)
- Erste Version 1990
- 1993 erste Tcl/Tk-Konferenz
- 1994 Ousterhout wechselt zu Sun, gründet Tcl-Abteilung
- 1997 ACM Software Award für Tcl/Tk
- 2012 Objekt-Orientierte Erweiterung

Syntax und Semantik

- Tcl-Programme (Skripte) bestehen aus einer Sequenz von **Kommandos** der Form

```
cmd arg1 arg2 arg3 ...
```

- Trennung der Kommandos durch Zeilenumbruch oder Semikolon
- Auswertung in drei Schritten:
  1. Gruppierung der Argumente (durch `{ ... }` oder `" ... "`)
  2. Substitution der Argumente
  3. Aufruf des Kommandos

Datentypen

- Strings
- **Keine** numerische Datentypen (aber es gibt ein Kommando dafür)
- Assoziativlisten (`array`, wie in Python) und Listen

Kommandos

- Variablenzuweisung: `set`
- Auswertung arithmetischer Ausdrücke: `expr`
- Auswertung eines Tcl-Kommandos: `eval`
- Fallunterscheidung und Schleifen:
  - `if`,
  - `while`, `foreach`, `for`, `break`, `continue`
  - `if`, `while` erwarten numerische Argumente (`0` ist `false`, wie in C)
- Ausnahmen: `try` und `catch`
- Arraymanipulation: `array`
- Ein/Ausgabe: `gets`, `puts`
- Fallunterscheidung: `switch`
  - Auf Werten oder **regulären Ausdrücken**

Substitution

- Variablen (brauchen nicht deklariert zu werden)

```
set x 5
puts {x is $x}
puts "x is $x"
```

- Kommandosubstitution: `[ ... ]` (kann geschachtelt werden)

```
set x [string length "foo"]
```

- Backslashes: \$, \n, \u001b, ...
- Gruppierung **vor** Substitution

Metaprogrammierung durch `eval`

- `eval` ruft den Tcl-Interpreter auf das Argument auf:

```
set cmd {puts {Hello, World!}}
...
eval $cmd
```

- Auswertung der Variablen zur **Ausführungszeit**:

```
set string "Hello, world!"
set cmd {puts $string}
unset string
eval $cmd
```

- Nutzung: **callbacks**, *e.g.* für GUI-Elemente

Beispiel

- Die Fakultätsfunktion, iterativ:

```
proc Factorial {x} {
  set i 1; set product 1
  while {$i <= $x} {
    set product [expr $product * $i]
    incr i
  }
  return $product
}
```


Beispiel

- Die Fakultätsfunktion, rekursiv:

```
proc Factorial {x} {
  if {$x <= 1} {
    return 1
  } else {
    return [expr $x * [Factorial [expr $x - 1]]]
  }
}
```


Die GUI-Bibliothek Tk

- Integraler Bestandteil der Sprache, und Grund für die Popularität
- Inzwischen etwas out-of-date, aber sehr stabil
- Grundprinzip:
  - GUI läuft asynchron
  - Erzeugt **Events**, an die **Callbacks** gebunden werden
- Eigene Bücherei, hat daher kein (kaum) natives Look&Feel
  - Sieht überall gleich (schlecht) aus

Tcl/Tk im Beispiel: Hello, World!

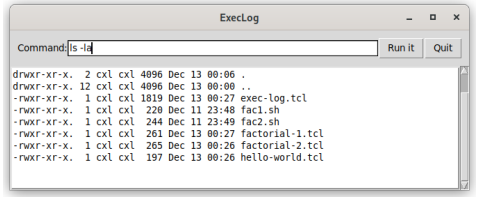
- Ein erstes Beispiel:

```
button .hello -text Hello \
              -command {puts stdout {Hello, World!}}
pack .hello -padx 20 -pady 10
```

- `button` deklariert Knopf
- `pack` plaziert Knopf im GUI

Ein längeres Beispiel

- `exec-log.tcl`



Steckbrief Tcl

Name	Tcl
Entstehung	Ab 1990 von John Ousterhout entwickelt
Programmierparadigma	Imperativ
Typen	Strings, Arrays, Listen
Typsystem	Ungetypt (schwach dynamisch getypt)
Parameterübergabe	call-by-name
Datenabstraktion	Wenig (lokale/global Variablen)
Anwendungsgebiet	"Glue language"
Besondere Kennzeichen	GUI-Bücherei Tk

Zusammenfassung

Zusammenfassung

- Skriptsprachen sind kein Programmierparadigma, eher eine Frage der Nutzpragmatik
- Verschiedene Kennzeichen von Skriptsprachen:
  - Kleiner Sprachkern, interpretiert
  - Schneller Entwicklungszyklus
  - Dient meist dazu, andere Programme zusammenzufügen
- Beispielsprachen: sh, Tcl