Programmier- und Unix-Propädeutika

Vom Shell-Befehl zum Programm

Carl von Ossietzky Universität Oldenburg
22. September 2026

Leonard Langer

Motivation

Gestern haben wir dem Computer **Befehle** gegeben.

Heute beschreiben wir ihm **Abläufe**.

Motivation: Von gestern zu heute

- Durch das Eingeben von Befehlen programmiert ihr den Computer etwas Bestimmtes zu tun
- Durch das Kombinieren von Befehlen schreibt ihr eigene kleine Programme

- Skripte erlauben es euch diese Programme wiederholbar abzuspeichern
- In Skripten könnt ihr aber auch Abläufe mit Daten und Entscheidungen beschreiben

```
echo "uni=https://uol.de" >> bookmarks.txt
grep '^uni=' bookmarks.txt | cut -d= -f2-
```

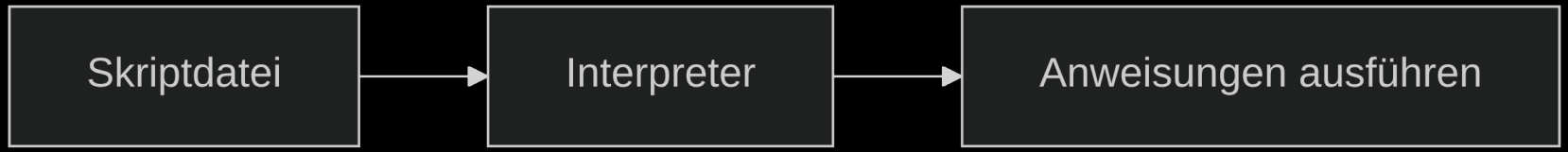
Konzepte

Was macht ein Programm aus?

Konzept: Skripte

```
#!/usr/bin/env bash

echo "Hello, world!"
```



Ein **Skript ist ein Programm** in einer Textdatei.

Konzept: Variablen & Datentypen

Variable

Ein Name für einen Wert

```
name  = "Ada"
alter = 18
```

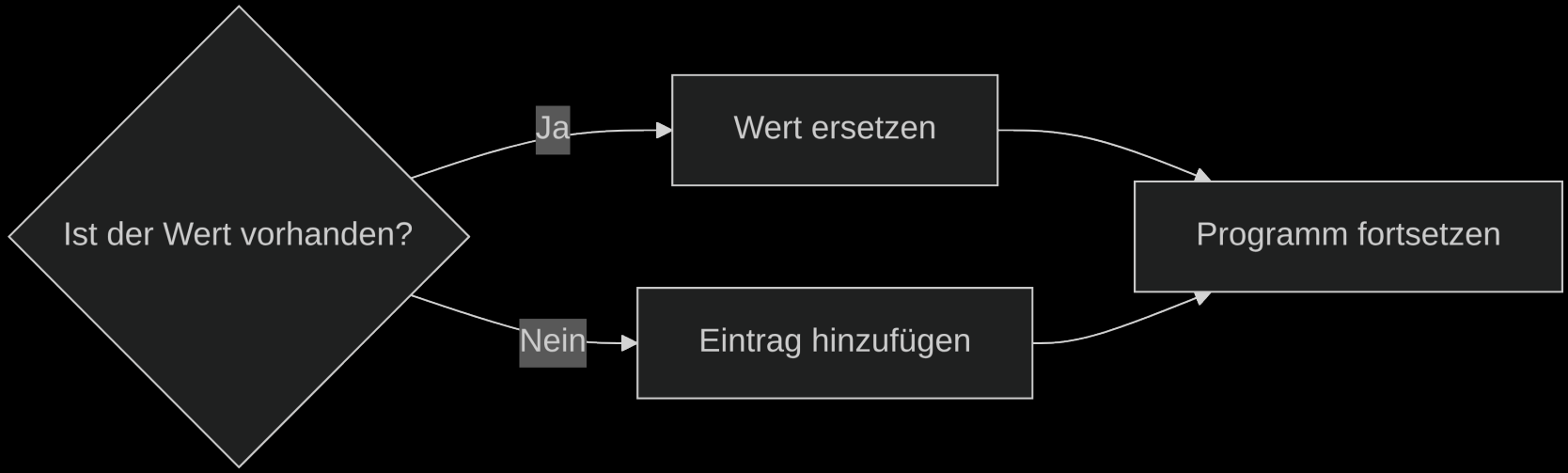
Datentyp

Bestimmt mögliche Operationen

- Text
- Zahl
- Wahrheitswert

Zuweisen | Lesen | Verändern

Konzept: Bedingungen (Kontrollfluss)



Eine `if`-Bedingung entscheidet, **welcher Weg** ausgeführt wird.

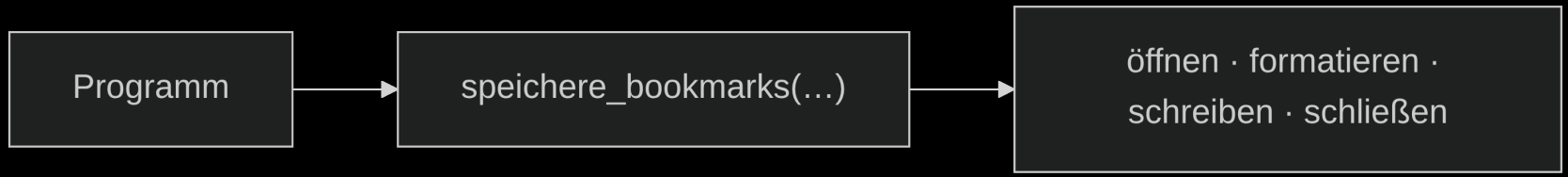
Konzept: Schleifen (Kontrollfluss)

```
für jedes element in sammlung:
    verarbeite element
```



Eine foreach-Schleife führt etwas **für jedes Element** aus.

Konzept: Abstraktion



Funktionen

- geben einer Teilaufgabe einen Namen
- können Eingaben erhalten und Ergebnisse liefern

Module & Imports

- teilen Funktionen zwischen mehreren Dateien

Bookmarks

Wir bauen unsere eigenen kleinen Programme!

Bookmarks: Was wollen wir bauen?

```
./push uni https://uol.de
./push mensa https://uol.de/mensa
./list
./get uni
./delete mensa
```

bookmarks.txt

```
uni=https://uol.de
mensa=https://uol.de/mensa
```

Bash

Nicht nur eine Shell!

Bash: Skripte

```
bookmarks/
├── bookmarks.txt # unsere Daten
├── bookmarks.sh  # geteilte Funktionen
├── push.sh       # erstelle/update bookmarks
├── list.sh       # liste alle bookmarks
├── get.sh        # erhalte den Wert einer bookmark
└── delete.sh     # lösche eine bookmark
```

```
#!/usr/bin/env bash

echo "Hello, Bookmarks!"
```

`chmod +x push` → `./push.sh`

Bash: Variablen

```
key="$1"
url="$2"
datei="bookmarks.txt"

printf '%s=%s\n' "$key" "$url" >> "$datei"
```

- Zuweisung: `name=wert`
- Lesen: `$name` oder `${name}`
- Argumente: `$1`, `$2`, …
- Variablen fast immer mit Anführungszeichen verwenden

Bash: Input

```
key="${1:-}"
url="${2:-}"

if [ -z "$key" ]; then
  read -r -p "Key: " key
fi

if [ -z "$url" ]; then
  read -r -p "URL: " url
fi
```

Input kann über **Argumente** oder **interaktiv** erfolgen.

Bash: Bedingungen (Kontrollfluss)

```
if grep -q "^${key}=" "$datei"; then
  grep -v "^${key}=" "$datei" > "${datei}.tmp"
  mv "${datei}.tmp" "$datei"
fi

printf '%s=%s\n' "$key" "$url" >> "$datei"
```

- `grep -q` prüft ohne Ausgabe
- Exit-Code `0` bedeutet Erfolg
- `if` entscheidet anhand des Exit-Codes

Bash: Schleifen (Kontrollfluss)

```
for key in $(cut -d= -f1 bookmarks.txt); do
  url=$(grep "^${key}=" bookmarks.txt | cut -d= -f2-)
  printf '%s → %s\n' "$key" "$url"
done
```

```
uni → https://uol.de
mensa → https://uol.de/mensa
```

Bash: Funktionen (Abstraktion)

```
bookmark_get() {
  grep "^${1}=" "$BOOKMARK_FILE" | cut -d= -f2-
}

bookmark_delete() {
  grep -v "^${1}=" "$BOOKMARK_FILE" > "${BOOKMARK_FILE}.tmp"
  mv "${BOOKMARK_FILE}.tmp" "$BOOKMARK_FILE"
}
```

`bookmark_get uni` statt der vollständigen Pipeline

Bash: `source` (Abstraktion)

`bookmarks.sh`

```
BOOKMARK_FILE="bookmarks.txt"

bookmark_get() { ...; }
bookmark_delete() { ...; }
bookmark_push() { ...; }
```

`get`

```
source ./bookmarks.sh
bookmark_get "$1"
```

Python

Neue Sprache, bekannte Konzepte.

Python: Skripte

```
#!/usr/bin/env python3

print("Hello, Bookmarks!")
```

```
python3 push.py

chmod +x push.py
./push.py
```

Datei → Interpreter → Anweisungen

Python: Variablen & Datentypen

```
key = "uni"
url = "https://uol.de"
gefunden = True
anzahl = 2
klein, gross = (0.01, 99999999.9)
```

Wert	Datentyp
`"uni"`	Text: `str`
`True`	Wahrheitswert: `bool`
`2`	Ganzzahl: `int`
`0.01`	Fließkommazahl: `float`

Kein `$` beim Lesen – Leerzeichen um `=` sind erlaubt.

Python: Dictionary (Komplexe Daten)

```python
bookmarks = {
    "uni": "https://uol.de",
    "mensa": "https://uol.de/mensa",
}

# lesen von Werten
print(bookmarks["uni"])

# schreiben von Werten
bookmarks["uni"] = "https://uol.de/studium"
```

- ordnet eindeutige **Schlüssel** ihren **Werten** zu
- heißt in Python `dict`

Python: Datei → Dictionary

```
def load_bookmarks():
    bookmarks = {}

    with open("bookmarks.txt", "r") as datei:
        for line in datei:
            key, url = line.rstrip("\n").split("=", 1)
            bookmarks[key] = url

    return bookmarks
```

`split("=", 1)` trennt nur am ersten Gleichheitszeichen.

Python: Kontrollfluss

```
bookmarks = load_bookmarks()

if key in bookmarks:
    print("Der Wert wird ersetzt.")

bookmarks[key] = url

for key, url in bookmarks.items():
    print(key + " → " + url)
```

Entscheiden und wiederholen – genau wie zuvor.

Python: Dictionary → Datei

```
def save_bookmarks(bookmarks):
    with open("bookmarks.txt", "w") as datei:
        for key, url in bookmarks.items():
            datei.write(key + "=" + url + "\n")
```

- `"w"` öffnet die Datei zum Schreiben
- die Schleife schreibt jeden Eintrag
- `with` schließt die Datei anschließend

Python: Funktionen (Abstraktion)

```
def push(key, url):
    bookmarks = load_bookmarks()
    bookmarks[key] = url
    save_bookmarks(bookmarks)

def get(key):
    bookmarks = load_bookmarks()
    return bookmarks[key]

def delete(key):
    bookmarks = load_bookmarks()
    del bookmarks[key]
    save_bookmarks(bookmarks)
```

Python: `import` (Abstraktion)

bookmarks.py

```
def push(key, url): ...
def get(key): ...
def delete(key): ...
```

get.py

```
import sys
from bookmarks import get

print(get(sys.argv[1]))
```

Bash & Python

Konzept	Bash	Python
Skript	`#!/usr/bin/env bash`	`#!/usr/bin/env python3`
Variable	`key="$1"`	`key = sys.argv[1]`
Bedingung	`if …; then`	`if …:`
Schleife	`for …; do`	`for …:`
Funktion	`name() { …; }`	`def name(…):`
Modul	`source datei`	`import modul`

Die Syntax ändert sich. Die Konzepte bleiben.

HILFE!!!!1!!11!! Help!

Was macht mein Programm da?!

Bei Unklarheiten …

- Ausgabe lesen
- Fehlerstelle suchen
- Werte zwischendurch ausgeben
- kleine Teile einzeln ausprobieren
- `man`, `help` und Python-REPL verwenden
- Andere fragen

⍰ Warning

Befehle und Programme aus dem Internet und von KI-Sprachmodellen sind oft sehr hilfreich. Doch seid euch sicher, dass ihr alles versteht, bevor ihr diese ausführt, da auch kleine Details oft große Auswirkungen haben können!

Danke

Diese Präsentation wurde …

- präsentiert mit
  - **presenterm** (https://mfontanini.github.io/presenterm/)
- erstellt mit
  - **helix** (https://helix-editor.com/)
- versioniert mit
  - **git** (https://git-scm.com/)
- dargestellt in
  - **kitty** (https://sw.kovidgoyal.net/kitty/)
- zur Verfügung gestellt von
  - **nixos** (https://nixos.org/)

… und mit viel Liebe gemacht <3