Python 3 - Grundlagen und Ökosystem

Unit 1: Einführung in Programmierung & Python-Tooling

Hussam Alafandi

January 10, 2026

Unit 1: Überblick

Ziele der Unit

- Verstehen, was Programmierung ist (Problem $\rightarrow$ Algorithmus $\rightarrow$ Code).
- Verstehen, wie Python Code ausführt (Interpreter, Skripte, Notebooks).
- Entwicklungsumgebungen kennenlernen: Jupyter und VS Code.
- Virtuelle Umgebungen erstellen und nutzen (venv, conda).
- Pakete installieren und Abhängigkeiten verwalten (pip, requirements.txt).
- Erste Python-Kommandos ausführen und Unterschiede Skript vs. Notebook kennen.

Agenda

1. Was ist Programmierung?
2. Wie funktioniert Python?
3. Skripte vs. Notebooks
4. Entwicklungsumgebungen (Jupyter, VS Code)
5. Virtuelle Umgebungen (venv, conda)
6. Pakete & Abhängigkeiten (pip, requirements.txt)
7. Praxis & Aufgaben

Einführung in Programmierung

Was ist Programmierung?

- Programmierung = Problemlösung durch präzise Anweisungen.
- Ein Programm ist eine Abfolge von Schritten, die ein Computer ausführt.
- Zentraler Ablauf:
  - Problem verstehen
  - Algorithmus entwerfen
  - Code implementieren
  - Testen & iterieren

Von Problem zu Code (Beispiel)

Problem: Berechne die Gesamtsumme eines Einkaufs.

Algorithmus:

1. Preise erfassen
2. Summe berechnen
3. Ergebnis ausgeben

Code: (wir starten heute nur mit sehr einfachen Beispielen)

Wie Python funktioniert

Wie führt Python Code aus?

- Python ist eine interpretierte Sprache: Code wird durch den **Interpreter** ausgeführt.
- Quellcode-Dateien enden meist auf `.py`.
- Fehler sind normal: Python gibt meist sehr hilfreiche Fehlermeldungen aus.
- Interaktiv vs. Datei-basiert:
  - Interaktiv: Python REPL (Shell)
  - Datei-basiert: Skripte (`python file.py`)

Python-Artefakte

- **Skript** (`.py`): Code wird von oben nach unten ausgeführt.
- **Notebook** (`.ipynb`): Code wird in Zellen ausgeführt (nicht zwingend linear).
- **Pakete**: Wiederverwendbarer Code, den wir installieren können.
- **Virtuelle Umgebung**: Isolierter Raum für Pakete pro Projekt.

Skripte vs. Notebooks

Skripte vs. Notebooks

- **Skripte:**
  - Gut für Softwareentwicklung und wiederholbare Abläufe
  - Saubere Projektstruktur
  - Einfach zu versionieren (Git)
- **Notebooks:**
  - Gut für Exploration, Lernen, Datenanalyse
  - Mischung aus Code, Text, Output
  - Gefahr: “Hidden State” durch nicht-lineare Ausführung

Entwicklungsumgebungen

Jupyter Notebook

- Interaktive Umgebung für Code + Text (Markdown) + Visualisierungen.
- Ideal für:
  - Lernen
  - Datenanalyse
  - Schnelles Prototyping
- Zellen:
  - Code-Zellen
  - Markdown-Zellen

VS Code (Warum es sich lohnt)

- Professioneller Editor für Skripte und Projekte.
- Vorteile:
  - Autocomplete & Linting
  - Debugging
  - Integriertes Terminal
  - Git-Integration
  - Notebook-Support

Praxis: Installation & Checks

Python Installation prüfen

- Version prüfen:

```
python --version
python3 --version
```

- Wo liegt der Interpreter?

```
which python      # Mac/Linux
where python      # Windows
```

Erste Python-Kommandos (REPL)

- Python Shell starten:

```
python
```

- Erste Befehle:

```
print("Hallo Python!")
2 + 3 * 4
```

- Python Shell verlassen:

```
exit()
```

Virtuelle Umgebungen

Warum virtuelle Umgebungen?

- Vermeidung von Versionskonflikten zwischen Projekten.
- Reproduzierbarkeit (gleiche Pakete $\rightarrow$ gleiche Ergebnisse).
- Saubere Abhängigkeiten pro Projekt.
- Best Practice in professionellen Teams.

Virtuelle Umgebung mit venv

Schritte:

1. Umgebung erstellen:

```
python -m venv .venv
```

2. Aktivieren:

```
source .venv/bin/activate      # Mac/Linux
.venv\Scripts\activate         # Windows
```

3. Deaktivieren:

```
deactivate
```

Virtuelle Umgebung mit Conda

Schritte:

1. Umgebung erstellen:

```
conda create --name py101 python=3.11
```

2. Aktivieren:

```
conda activate py101
```

3. Deaktivieren:

```
conda deactivate
```

venv vs. Conda (Kurzvergleich)

- venv:
  - In Python eingebaut (kein extra Tool nötig)
  - Sehr gut für “pure Python” Projekte
- Conda:
  - Stärker im wissenschaftlichen Umfeld (native Libraries)
  - Separater Paketmanager + Environment Manager

Pakete & Abhängigkeiten

Standard Library vs. Third-Party

- **Standard Library:** Kommt mit Python (z. B. `math`, `csv`, `json`).
- **Third-Party Packages:** Müssen installiert werden (z. B. `numpy`, `pandas`).
- Warum das wichtig ist:
  - Projekte brauchen reproduzierbare Abhängigkeiten
  - Teamarbeit erfordert gleiche Setups

Pakete installieren mit pip

- Paket installieren:

```
pip install numpy pandas
```

- Installierte Pakete anzeigen:

```
pip list
```

- Abhängigkeiten exportieren:

```
pip freeze > requirements.txt
```

- Abhängigkeiten installieren:

```
pip install -r requirements.txt
```

Praxis: Skript vs. Notebook

Einfaches Skript: `hello.py`

```
1 print("Hallo aus einem Skript!")
2 x = 2 + 3 * 4
3 print("Ergebnis:", x)
```

Notebook: gleiche Logik, andere Arbeitsweise

- Im Notebook:
  - Code in Zellen ausführen
  - Markdown für Dokumentation nutzen
  - Output direkt unter der Zelle
- Wichtig:
  - Notebooks können “State” behalten
  - Zellen sollten sinnvoll und reproduzierbar ausgeführt werden

Aufgaben (Unit 1)

Aufgabe 1: Setup & Dokumentation

- Python-Version prüfen und dokumentieren.
- Virtuelle Umgebung erstellen und aktivieren.
- `numpy` installieren.
- Schritte in `environment_setup.md` dokumentieren.

Aufgabe 2: Skript vs. Notebook

- Erstelle `hello.py` und `hello.ipynb` mit:
  - Ausgabe einer Begrüßung
  - Einer einfachen Berechnung
- Führe beide aus und vergleiche den Ablauf.

Aufgabe 3: Kurzreflexion

- Beantworte kurz:
  1. Was macht der Python Interpreter?
  2. Wann ist ein Notebook sinnvoller als ein Skript?
  3. Warum sind virtuelle Umgebungen nützlich?

Fragen?