



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Chapter 01

MODERN PYTHON DEVELOPMENT

Where We Are Now

Congratulations! In Part 1, you achieved:

- ✓ Git & GitHub setup with SSH authentication
- ✓ VS Code configured with essential extensions
- ✓ GitHub Copilot activated for AI assistance
- ✓ First repository created and pushed

Today: Transform from script writers to professional Python developers!

Today's Learning Objectives

By the end of this lecture, you will:

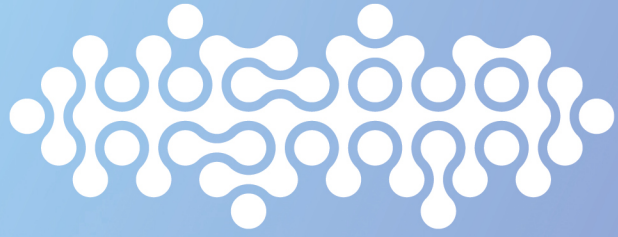
1.  **Understand** why Python dominates modern software development
2.  **Distinguish** script Python from production Python
3.  **Set up** your first professional Python project
4.  **Master** `uv` - the revolutionary Python package manager
5.  **Understand** virtual environments and `pyproject.toml`
6.  **Experience** the modern Python development workflow

Quick Poll: Your Python Experience

Raise your hand if you have...

- Used Python before (any context)?
- Written more than 100 lines of Python?
- Used `pip install` to add packages?
- Created a virtual environment?
- Published a Python package?

Today: We'll level up your Python skills regardless of where you start!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Why Python?

1

Why Python? The Numbers

GitHub (2024-2025):

- **19.8 million** repositories use Python
- **2nd most popular** language (after JavaScript)

Stack Overflow Developer Survey 2024:

- **51%** of professional developers use Python
- **#1 language** developers want to learn

TIOBE Index (2025):

- **#1 programming language** globally
- **15.2%** market share

Why Python Wins

Readability - Code that reads like English:

```
def calculate_average(numbers):  
    """Calculate the average of a list of numbers."""  
    return sum(numbers) / len(numbers)  
  
students = [85, 92, 78, 90, 88]  
average = calculate_average(students)  
print(f"Average grade: {average}")
```

Compare this to Java or C++ - Python wins on clarity!

Python: One Language, Every Domain

Development:

- Web: Django, Flask, FastAPI
- Desktop: PyQt, Tkinter
- Games: Pygame

Data & AI:

- Data Science: Pandas, NumPy
- ML: TensorFlow, PyTorch
- AI: LangChain, OpenAI SDK

Career Impact: Learn Python once, work in any domain.

The AI Revolution is Python

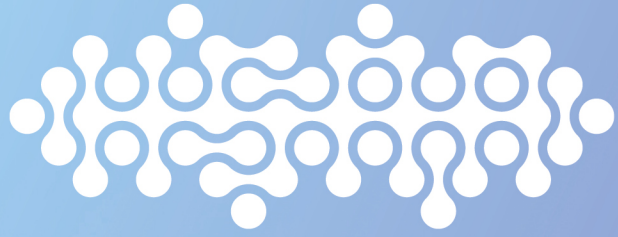
We're living through an AI revolution, and Python is the language of AI:

- **PyTorch & TensorFlow** - The dominant ML frameworks
- **Hugging Face** - The AI model hub (primarily Python)
- **LangChain, LlamaIndex** - LLM application frameworks
- **OpenAI API, Anthropic API** - Python SDKs are first-class

If you want to work with cutting-edge AI, Python isn't optional.

The Bottom Line: Why Python

1.  **Ubiquitous** - Used everywhere, from startups to NASA
2.  **Productive** - Write less code, accomplish more
3.  **Versatile** - One language for web, data, AI, automation
4.  **In-demand** - Top salaries, most job openings
5.  **Future-proof** - Leading AI/ML language
6.  **Beginner-friendly** - Learn concepts, not syntax quirks
7.  **Professional-grade** - Scales from scripts to enterprise



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Script vs. Production Python

2

The Two Pythons

Script Python (What tutorials teach):

```
# analysis.py
import pandas as pd
data = pd.read_csv("data.csv")
result = data.describe()
print(result)
```

```
python analysis.py # Works... for now
```

Question: What happens when your colleague tries to run this?

Production Python: The Professional Way

```
# src/analytics/pipeline.py
import pandas as pd
from pathlib import Path

def analyze_data(file_path: Path) -> pd.DataFrame:
    """Analyze data from CSV file and return statistics."""
    data = pd.read_csv(file_path)
    return data.describe()
```

```
# Modern uv workflow - proper packaging
uv run analytics-cli analyze data.csv
```

Same functionality, professional structure!

What Changes at Scale?

Problems:

- "It works on my machine"
- How to distribute code?
- Bugs in production cost money
- 10 developers, same codebase

Solutions:

- Reproducible environments
- Build & publish packages
- Testing, linting, type checking
- Standardized project structure

Today's focus: Reproducible environments with `uv`

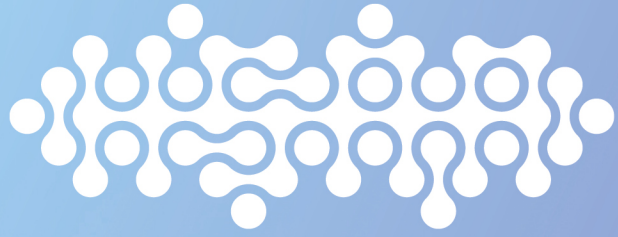
Discussion: "It Works on My Machine"

Scenario: You write a Python script that works perfectly. You send it to a colleague. It crashes.

What could go wrong?

- Different Python version?
- Missing packages?
- Different package versions?
- Operating system differences?

How would you solve this?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Your First Professional Project

3

Hands-On: Your First Professional Project

We'll transform script Python into production Python!

What you'll do:

1. Download starter code (Road Profile Viewer)
2. Set up your local environment
3. Initialize Git repository
4. Connect to GitHub
5. Run with modern `uv` tooling

Let's begin!

Step 1: Download the Code

Course students:

1. Go to Canvas course page
2. Download `road-profile-viewer.7z`
3. Extract the archive

External users:

1. Visit GitHub release page
2. Download source code archive
3. Extract

What you'll find: Python scripts for visualizing road elevation profiles

Wait - Why a ZIP File?

"Didn't we just say Git is better than zipped files?"

Yes! And you're right to notice.

Why we're doing this:

1. Start with a zip file (the "old way")
2. Initialize Git yourself
3. Create your own GitHub repository
4. Connect and push

Doing it once yourself teaches you how repositories are born.

Think of it as training wheels before the Tour de France!

Step 2-3: Setup and Customize

```
# Create project folder
mkdir road-profile-viewer
cd road-profile-viewer
# Copy extracted files here, then open in VS Code
code .
```

Edit `pyproject.toml` :

```
# Change this line to YOUR info:
authors = [{name = "YOUR_GITHUB_USERNAME", email = "your@email.com"}]
```

Save the file! This makes you the official author.

Step 4: Initialize Git

```
# Initialize Git in your project folder
git init

# Add all files to staging
git add .

# Create your initial commit
git commit -m "Initial commit: Road profile viewer scripts"

# Check your status
git status
```

Checkpoint: You now have a local Git repository with your first commit!

Step 5-6: Connect to GitHub

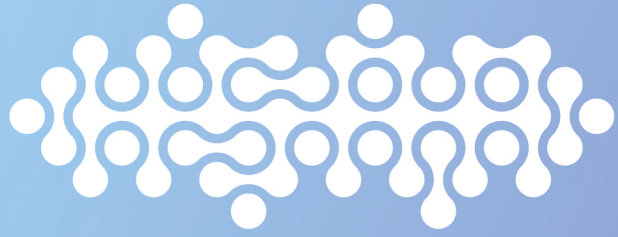
On GitHub:

1. Click "+" → "New repository"
2. Name: `road-profile-viewer`
3. **Do NOT** initialize with README or .gitignore

In terminal:

```
git remote add origin https://github.com/YOUR_USERNAME/road-profile-viewer.git  
git branch -M main  
git push -u origin main
```

Refresh GitHub - your code is there!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Modern Python with uv

4

Install uv: The Modern Way

Windows (PowerShell):

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

Ubuntu/Linux:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Verify:

```
uv --version  
# Should show uv 0.5.x or higher
```

Run Your Application!

```
cd src/road-profile-viewer  
uv run main.py
```

That's it! `uv` automatically:

-  Detects Python version & installs dependencies
-  Creates isolated environment
-  Starts your application

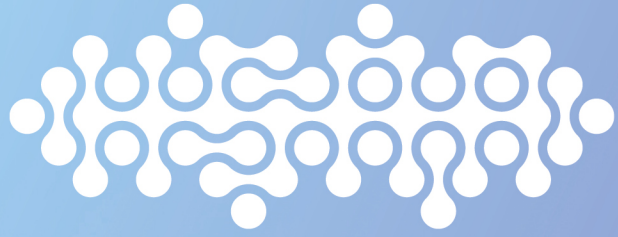
Open browser: `http://127.0.0.1:8050`

What Just Happened?

With a single command (`uv run main.py`), you:

- Installed the correct Python version automatically
- Set up all project dependencies (Dash, Plotly, Pandas, etc.)
- Created an isolated environment (no conflicts!)
- Started a web server with interactive visualizations

This is the modern Python workflow - no manual setup, no "it works on my machine" problems!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Virtual Environments & Dependencies

5

The Problem: Dependency Chaos

Imagine working on three projects:

- **Project A:** Requires `pandas 1.5.0`
- **Project B:** Requires `pandas 2.1.0`
- **Your System:** Can only have ONE global pandas!

```
pip install pandas==1.5.0 # For Project A  
pip install pandas==2.1.0 # Overwrites! Project A breaks!
```

This is "dependency hell" - and it's why virtual environments exist.

Virtual Environments: The Solution

Think of apartments in a building:

- Each apartment (virtual environment) has its own furniture (packages)
- Residents (projects) don't interfere with each other
- The building (your computer) hosts many independent apartments
- Each can have different versions of the same furniture

Virtual environment = Isolated Python installation for each project

Inside a Virtual Environment

```
.venv/                                # Virtual environment folder
├── bin/ (or Scripts/ on Windows)
│   ├── python                        # Python executable
│   ├── pip                          # Package installer
│   └── activate                      # Activation script
├── lib/
│   └── python3.12/
│       └── site-packages/           # YOUR PACKAGES LIVE HERE
│           ├── pandas/
│           ├── dash/
│           └── plotly/
└── pyvenv.cfg
```

Key: Packages go into THIS environment, not your global Python!

The Old Way: Manual Everything

```
python -m venv .venv          # 1. Create (slow!)  
.venv\Scripts\activate        # 2. Activate (OS-specific!)  
pip install --upgrade pip     # 3. Upgrade pip  
pip install dash plotly pandas # 4. Install deps  
python main.py                # 5. Finally run!
```

5+ steps, OS-specific commands, error-prone, slow...

The New Way: uv Does It All

```
uv run main.py # That's it. Seriously.
```

What uv does automatically:

```
[1] Read pyproject.toml
    ↓
[2] Create .venv/ (in milliseconds!)
    ↓
[3] Resolve & lock dependencies
    ↓
[4] Install only what's missing (cached!)
    ↓
[5] Run your script
```

One command replaces six manual steps!

Why uv is Revolutionary

Speed Comparison (venv + installing 10 packages):

Tool	Time
Traditional (<code>venv</code> + <code>pip</code>)	~30 seconds
<code>uv</code>	~1 second

Why? uv is written in Rust (compiled, parallel, optimized)

Traditional tools: Python-based, slow, no parallelism

What Makes uv So Fast?

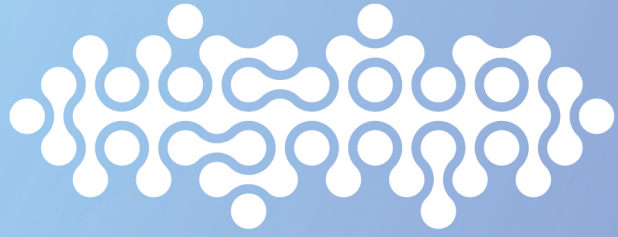
Traditional (pip, poetry):

- Written in Python
- New process for everything
- No parallel operations
- Inefficient file I/O

uv (Rust):

- Compiled to native code
- Parallel downloads
- Optimized file operations
- Global package cache

Result: 10-100x faster than traditional tools!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Understanding pyproject.toml

6

Understanding pyproject.toml

The single source of truth for your Python project:

```
[project]
name = "road-profile-viewer"
version = "0.1.0"
requires-python = ">=3.10"

dependencies = [
    "dash>=2.14.0",
    "plotly>=5.18.0",
    "pandas>=2.1.0",
]

[project.optional-dependencies]
dev = ["pytest>=7.4.0", "ruff>=0.1.0"]
```

Everything in one file: Metadata + dependencies + build config

Version Specifiers Explained

```
dependencies = [  
    "dash>=2.14.0",      # Any version 2.14.0 or higher  
    "plotly~=5.18.0",    # 5.18.x only (patch updates OK)  
    "pandas>=2.1,<3.0",  # 2.1+ but not 3.x  
    "numpy==1.24.0",     # Exact version (usually too strict!)  
]
```

Best practice: Use `>=` for most dependencies.

Let `uv` resolve compatible versions automatically!

Three Ways to Run Your App

Method	Command	Best For
uv run	<code>uv run main.py</code>	Quick development
Manual venv	<code>python main.py</code>	Debugging, IDE setup
Named script	<code>road-viewer</code>	Production, distribution

For daily development: Use `uv run` - it's fastest and simplest!

Hands-On: Add a Named Script

Add to your `pyproject.toml`:

```
[project.scripts]
road-viewer = "main:main"
```

Update `main.py` with a main function:

```
def main():
    """Main entry point for Road Profile Viewer."""
    app = create_app()
    app.run_server(debug=True)

if __name__ == "__main__":
    main()
```

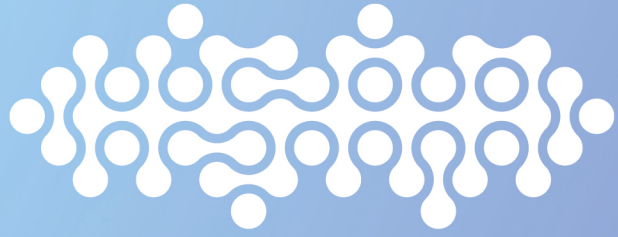
Run Your Named Script

```
# Sync your project (registers the script)
uv sync

# Now run with the clean command!
uv run road-viewer
```

You've created a professional CLI command!

Real-world examples: `pytest`, `black`, `ruff`, `uvicorn` - all use this pattern!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

uv Command Reference

7

Quick Reference: Essential uv Commands

```
# Running code
uv run script.py          # Run a Python script
uv run command            # Run a named script

# Package management (modern way)
uv add package            # Add package to project
uv add package --dev      # Add as dev dependency
uv sync                  # Sync environment with pyproject.toml

# Environment
uv venv                  # Create virtual environment
uv venv --python 3.12    # With specific Python version
```

More uv Commands

```
# Project initialization
uv init                # Create new Python project
uv init --app          # Create application project

# Python version management
uv python install 3.12  # Install Python 3.12
uv python list         # List available versions

# Global tools
uv tool install ruff    # Install tool globally
uv tool list           # List global tools
```

Key: `uv add` updates `pyproject.toml` automatically!

The Complete Picture

```
You type: uv run main.py
      ↓
    [uv reads pyproject.toml]
      ↓
  [Checks/creates .venv/]
      ↓
[Resolves dependencies - all compatible?]
      ↓
[Installs from cache or downloads]
      ↓
  [Runs your script]
      ↓
Your app runs!
```

All this complexity is hidden behind one command!

Key Takeaways: Virtual Environments

1. Virtual Environments:

- Isolated spaces for project dependencies
- Prevent conflicts between projects
- Located in `.venv/` folder

2. `pyproject.toml`:

- Single source of truth for your project
- Replaces `requirements.txt` + `setup.py`
- Modern Python standard

Key Takeaways: Why uv Matters

3. Why uv:

- **Speed:** 10-100x faster than traditional tools
- **Simplicity:** One command replaces many
- **Reliability:** Proper dependency resolution
- **Modern:** Built for 2025's workflows

4. Professional Workflow:

```
uv run main.py # vs 6 manual steps with pip/venv
```

The future of Python development is here!

Test Your Understanding

Ready to assess your knowledge?

- **20 questions** covering all topics
- **Instant feedback** on your answers
- **Not graded** - purely for self-assessment
- Questions similar to these **will appear in the exam**

Take the Chapter 01 (Modern Python Development) Quiz:

<https://forms.gle/WwnX3ytHAd53sZY39>

Summary: What You Learned Today

1. **Python dominates** - #1 language, essential for AI/ML
2. **Script vs Production** - Professional code needs structure
3. **Virtual environments** - Isolated dependencies per project
4. **pyproject.toml** - Single source of truth
5. **uv revolution** - 10-100x faster, one command does it all
6. **Modern workflow** - `uv run` replaces 6 manual steps

What's Next?

Chapter 02: Code Quality in Practice

- PEP 8 compliance with Ruff
- Linting and formatting
- Type hints and type checking
- Pre-commit hooks

Your homework: Complete the Road Profile Viewer setup and take the quiz!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Questions?

THANK YOU