MODULE 1: CORE PYTHON & DATA

WEEK: 3 LECTURE: 12

DATE: 03/09/2025

INSTRUCTOR: HAMZA SAJID

THE DEVELOPER'S SANDBOX: MASTERING VIRTUAL ENVIRONMENTS

Welcome back! We've learned to use powerful modules, but a new problem arises: how do we manage these external tools? What happens if one project needs an old version of a library, and another project needs a new version? Today, we learn the single most important practice for professional Python development: **virtual environments**.

TODAY'S AGENDA

- **The "Why" & The Environment Lifecycle**
  - The Global Mess & The Dependency Problem
  - The Solution: Isolated Environments with Conda
  - The Basic Conda Workflow: Create, Activate, Deactivate
  - **Advanced Environment Management: List, Clone, Rename, Remove**
  - *Interactive Session: Creating and Managing Your First Conda Environment*
- **The "How" - Managing Dependencies**
  - **conda install vs. pip install: Which to Use?**
  - **Reproducibility: The environment.yml File**
  - **Live Demo / In-Class Exercise: The Power of Isolation with Conda**
- **Practical Examples & The Hands-On Lab**
  - Real-World Scenarios: Web Dev, Data Science, AI
  - **Conda Environment Management Cheat Sheet**
  - **Hands-On Lab: Project Setup with requests**
  - Q&A and Wrap-up

THE "WHY" - THE DEPENDENCY PROBLEM
THE GLOBAL MESS

- When you first install Python, there is one central location where all third-party packages are installed. This is called the **global site-packages directory**.
- If you just type pip install <package>, it goes into this global space.
- **The Problem:** Imagine you have two projects:
  - **Project A** is an old, stable project that relies on Report-Generator v1.2.
  - **Project B** is a new project you're starting, and you want to use the new features in Report-Generator v2.0.
- If you install v2.0 for Project B, you might accidentally break Project A! This is called **Dependency Hell**, and it's a nightmare for developers.

THE SOLUTION: ISOLATED ENVIRONMENTS

- A **virtual environment** is a self-contained directory that holds a particular version of Python plus all the packages that a specific project needs. It's like giving each project its own private, isolated workshop.
- **Analogy:** Think of it as giving each project its own private, isolated workshop. Project A gets a workshop with all its old, trusted tools. Project B gets a brand-new workshop where it can install the latest and greatest tools. The tools in one workshop don't affect the other.

CONDA (VIA MINICONDA/ANACONDA):

- **What it is:** A powerful **package and environment manager**.
- **Key Advantages:**
  - **Manages Python Versions:** Conda can install different versions of Python itself (e.g., 3.8, 3.9, 3.10) in different environments. This is its killer feature.
  - **Manages Non-Python Dependencies:** Crucial for AI/Data Science, as it can install complex libraries like CUDA and MKL that Python packages depend on.
  - **Robust Dependency Resolution:** It has a powerful solver to handle complex package dependencies.

VENV (THE BUILT-IN ALTERNATIVE):

- **What it is:** A lightweight environment manager included with Python.
- **How it's different:** It can only create environments with the *same Python version* used to create them. It cannot manage non-Python packages. It's great for simple, pure-Python web applications, but less suited for complex scientific computing.

THE BASIC CONDA WORKFLOW

- Let's learn the essential commands.
- **Create an environment:** You give it a name and specify the Python version.
  conda create --name my-first-env python=3.9
- **Activate the environment:** This "enters" the isolated workshop.
  conda activate my-first-env
  (Your terminal prompt will change to show (my-first-env))
- **Deactivate the environment:** This exits the workshop.
  conda deactivate
- **List all environments:**
  conda env list
- **Remove an environment:**
  conda env remove --name my-first-env

CLONING (DUPLICATING) ENVIRONMENTS

- This is an incredibly useful feature. Cloning creates an exact copy of an existing environment under a new name.
- **Why do this?** Imagine you have a stable, working environment for a project (my_project_stable). You want to test a new, potentially breaking version of a library. Instead of risking your stable environment, you clone it!

- **Clone command:**
  conda create --name <new_env_name> --clone <source_env_name>
- **Example Workflow:**
  - conda create --name my_project_testing --clone my_project_stable
  - conda activate my_project_testing
  - conda install "pandas>2.0" *(...do your testing...)*
  - If it breaks, no problem! Just remove the testing environment:
    conda env remove --name my_project_testing

THE "HOW" - MANAGING DEPENDENCIES WITH CONDA CONDA INSTALL VS. PIP INSTALL

- Inside a Conda environment, you can use both. Here's the rule of thumb:
  - **Always try conda install <package> first.** This uses Anaconda's channels, which contain tested, pre-compiled packages that work well together.
  - If a package is not available on Conda channels, then use pip install <package>. pip will install from the Python Package Index (PyPI).
- **Reproducibility: The environment.yml File**
- This is the Conda equivalent of requirements.txt, but it's more powerful. It saves the environment's name, channels, Python version, and all package dependencies.
  - **To save your environment:**
    conda env export > environment.yml
  - **To recreate an environment from the file:**
    conda env create -f environment.yml

LIVE DEMO / IN-CLASS EXERCISE: THE POWER OF ISOLATION WITH CONDA

- **Step 1: Create Project "Old"**
- Create a directory project_old and cd into it.
- Create a Conda environment with an older Python version:
  conda create --name old_flask_project python=3.8
- **Activate it:** conda activate old_flask_project
- Install an old version of Flask: conda install "flask<2.0"
- Check the installed packages: conda list flask (You'll see Flask version 1.x.x).
- Create a file app_old.py with this code:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")

def hello(): return "Hello from the OLD Flask project on Python 3.8!"
```

STEP 2: CREATE PROJECT "NEW"

- Go back to your parent directory and create project_new. cd into it.
- Create a new environment with a modern Python version:
  conda create --name new_flask_project python=3.10
- **Activate it:** conda activate new_flask_project
- Install a new version of Flask: conda install "flask>=2.0"
- Check the packages: conda list flask (You'll see Flask version 2.x.x or higher).
- Create a file app_new.py with this code using a modern feature:

```
from flask import Flask

app = Flask(__name__)

@app.get("/")

def hello(): return "Hello from the NEW Flask project on Python 3.10!"
```

- **The "Aha!" Moment:** You now have two projects on your computer with different Python versions *and* conflicting library versions, but they are perfectly isolated and won't interfere with each other.

CONDA ENVIRONMENT MANAGEMENT CHEAT SHEET

Task	Command
Create a new environment	conda create --name <env_name> python=3.9 <packages...>
Activate an environment	conda activate <env_name>
Deactivate an environment	conda deactivate
List all environments	conda env list
Clone an environment	conda create --name <new_name> --clone <source_name>
Remove an environment	conda env remove --name <env_name>
Export environment to file	conda env export > environment.yml
Create environment from file	conda env create -f environment.yml

HANDS-ON LAB: PROJECT SETUP WITH REQUESTS

- **Goal:** To go through the full, professional Conda workflow: create a project, set up an environment, install a package, write a script, and generate an environment.yml file.
- **The Project:** We will write a simple script that uses the requests library to fetch data from a public API.

STEP-BY-STEP INSTRUCTIONS:

- **Create the Project Directory:**
  - Create a new folder called api_project.
  - Navigate into this folder in your terminal.
- **Create and Activate the Conda Environment:**
  - Run this single command to create the environment and install requests at the same time: conda create --name api_project python=3.9 requests
  - When it's done, activate it: conda activate api_project
- **Write the Python Script:**
  - Create a new file named fetch_data.py.
  - Write the following code:

- **Run Your Script:**
  - In your terminal (with the env active), run: python fetch_data.py
- **Generate the environment.yml File:**
  - This is the final, crucial step for reproducibility.
  - Run: conda env export > environment.yml
  - Open the environment.yml file. Notice how it contains much more information than a requirements.txt file, including the Python version and all dependencies.