BA Course "6,243 Introduction to Programming"

January, 2026

Overview

Learning Objectives: After completion of the course, students will be able to design and execute projects in Economics and Econometrics in a systematic and reproducible way. The course has two core objectives: to develop intermediate programming skills and to apply collaborative, reproducible code development practices. First, students will understand and apply essential programming practices in Python, including writing modular code, documenting functions and implementing error handling. They will also be able to use object oriented programming by creating their own classes to structure larger projects. Second, students will be able to collaborate on projects using Git and version control, manage shared code and work with branching and pull requests. They will know how to design relational database schemas and interact with relational data from Python. They will also understand central concepts of DevOps, including continuous integration workflows and automated testing. Throughout the course they will learn to produce code that is reusable, maintainable and easy to share.

Philosophy of the course: The course aims to teach and coach young economists to code and work in a collaborative, structured and impactful way. The focus is on both traditional teaching and practical work, with guided exercises and project coaching in settings that reflect real work in research and industry.

Target audience: The course is mandatory for students in the Bachelor program in Economics and is offered in the second semester. Typically, participants have completed Data Handling and Statistics in the previous semester and are taking Introductory Econometrics in parallel. They already have solid working knowledge of R, including basic data manipulation and exploratory analysis.

Structure of the course:
- 12 units of 90′ for the main lecture;
- 10 units of 90′ for the exercises (week 1, 2, 3, 4, 6, 8, 9, 10, 11, 12);

Prerequisites:
- Elementary programming skills (see syllabus "Data Handling" course)
- Knowledge of basic Data structures (see syllabus "Data Handling" course)
- Statistics

Examination

The student assessment consists of two main components:

1. **Final Exam (30% of the grade)**: This exam will cover the core theoretical concepts of the course. Form: digital exam, BYOD, 60′. Multiple-choice questions.
2. **Group Project (70% of the grade)**: This is the primary applied assessment. Students will work in groups of 4-5 to design and build a 'software tool' that applies the skills learned in the course.

The central goal of the project is for students to develop a custom Python class that is useful for a specific economic analysis. The emphasis is on the technical implementation, not the depth of the economic analysis. This could involve, for example, handling a unique data source, automating common analytical tasks, displaying results in a novel way, etc. We'll provide general ideas on how classes can be used throughout the course.

Tutorials

- The course includes 10 tutorial sessions to accompany the lectures.
- A team of 4 Teaching Assistants (TAs) working on each exercise session.
- First 8 tutorials: (Skill Building)
  - These are classic, hands-on sessions for students to practice the concepts from the lectures.
  - They will be held as a single, combined session for all students. Or
  - 1 TA will lead the instruction, while the other 3 TAs provide hands-on support and answer individual questions.
- Final 2 tutorials: (Project Support)
  - These sessions are structured as "office hours" for the group projects.
  - Purpose: This time is dedicated for groups to meet with TAs to get help, troubleshoot problems, and receive direct guidance on their project.

Detailed Lecture Plan

Lecture 1	The big picture. Set up your environment.	Aurélien & Franzi
Lecture 2	Working together: intro to version control and git	Aurélien
Lecture 3	Working together: more about version control and git	Aurélien
Lecture 4	Intro to python	Franzi
Lecture 5	Introduction to Classes (OOP)	Franzi
Lecture 6	Python for Data: Pandas	Franzi
Break		
Lecture 7	Error Handling, documentation, Debugging	Franzi
Lecture 8	Short overview of databases and relational database management systems	Aurélien
Lecture 9	Devops, Continuous Integration	Aurélien
Lecture 10	Guest Lecture	Guest
Lecture 11	Case Study: Building a Class for Geodata	Aurélien & Franzi
Lecture 12	Conclusion	Aurélien & Franzi
Optional alternatives (instead of 8 or 9)		
	Communication: dashboards / matplotlib	
	Machine Learning and basic predictions with PyTorch	

Lecture 1: The Big Picture. Set up your environment *F, A*

Lecture 1 defines the objectives of the course and motivates why programming and programming in a shareable, reproducible way is crucial for Economists. Lecture 1 also introduces students to Python and shows them how to set up their working environment.

Students will install python, learn about tooling and package management with UV (as a replacement for pip), install and work with Visual Studio Code, and CLI (Command-Line Interface) to install everything. Students also learn about how to organize files and write file names correctly.

Exercise week 1: Students will have time to set up their environment and be ready for the lecture. They will also install git and create a github account.

Lecture 2-3: Working Together: Intro to Version Control and Git *A*

The lecture introduces version control as a central element of collaborative data work. It covers the main Git concepts such as repositories, commits, staging, branching, and merging. Students learn how to create and configure a local repository and how to connect a project to GitHub. The session discusses branching strategies for small teams, the use of pull requests, and practical conflict resolu-

tion. It also addresses project organization with clear folder structures and naming conventions. It then covers Github and how to work with a remote repository.

The lecture includes a hands on part in which students create a repository, push and pull changes, work with branches, merge them and resolve conflicts.

Exercise weeks 2-3: Students will have time to experiment with git and github.

Sources:
- "Introduction to git" from Jam Simson
- "Git Version Control", "Research Software Engineering", Matt Bannert

Lecture 4: Intro to Python *F*

Lecture 4 provides the essential foundation for the course. This lecture will cover Python's core syntax, its fundamental data types (like strings and integers), and data structures (like lists, dictionaries, ...). We will also walk through control flow (loops and conditionals) and demonstrate how to write functions. Throughout the lecture, we will highlight the differences between R and Python (such as 0-indexing and indentation) to help students adapt their existing programming knowledge.

Exercise week 4: Students apply basic concepts in Python learned in the course.

Lecture 5: Introduction to Classes (OOP) *F*

This lecture introduces students to the fundamental concepts of Object-Oriented Programming (OOP). We will explain how OOP is used to structure code by bundling data and functionality into reusable blueprints. Students will learn the complete process of how to define their own custom class, covering the essential components: writing the constructor, defining attributes, and creating methods. The goal is to ensure students can confidently build and use their own custom objects in Python. After covering these basics, we will explore some practical examples to showcase what can be done with a class, helping to spark ideas for the final group projects.

Lecture 6: Python for Data *F*

This lecture introduces students to Pandas, along with some NumPy essentials. We will cover the workflow for data handling: reading data into a DataFrame, inspecting its structure, and managing missing data. Students will then learn the core tools for data manipulation, including filtering rows, grouping data, aggregating results, and merging different datasets. The goal is to show how to perform basic (descriptive) analysis using interesting, economics-related data. Finally, we will introduce basic visualization tools, demonstrating how to generate simple plots using matplotlib to visually inspect results.

Optional: Time permitting, we may also provide a brief outlook on modern high-performance alternatives like Polars or tensor-based frameworks like PyTorch.

Exercise week 6: Students apply classes and work with data in Python.

Lecture 7: Error Handling, Documentation *F*

This lecture focuses on the critical skills needed to transform a basic script into a reliable and shareable tool. We will cover the principles of Error Handling, teaching students how to anticipate common problems (like bad data inputs) and provide clear, helpful feedback to the user. The second half focuses on Documentation, explaining how to write clear descriptions for functions and classes. The goal is to ensure that the tools built in the group projects are robust, easy to use, and understandable by others.

Lecture 8: Short overview of databases and relational database management systems (to drop if too much) *A*

This lecture introduces students to databases and relational database management systems. Using PostgreSQL, students learn what are database systems in order to manage and process large data sets. They will be introduced to creating, manipulating, and querying databases using SQL.

Exercise weeks 7-8: Students run simple sql queries and set up a basic relational database management system.

Sources:
- "Big Data Analytics" from Ulrich Matter
- "Databases", "Research Software Engineering", Matt Bannert

Lecture 9: DevOps (to drop if too much) *A*

This lecture introduces students to basic principles of DevOps, like containerization and docker, Continuous Integration - Continuous Development (CI/CD). As an example, we'll run a simple API in docker and access the database in a container with postgres, or we will use GitHub Actions to update the api automatically.

Exercise weeks 9-10: tbd.

Sources:
- Survey with a Database Backend
- "Github Action Demo", Matt Bannert

Lecture 10: Guest Lecture

Lecture 10 will be a guest lecture. The idea is to invite a speaker from industry or academia who works in a field where the programming and data skills taught in this course are highly valuable. This lecture will provide students with a practical, real-world perspective, as the speaker will explain how these concepts are applied in their day-to-day work.

Lecture 11: Case Study: Building a Class for Geodata *F, A*

Lecture 11 is a practical, application-focused session that demonstrates how to combine several of the tools learned in the course. We will walk through the process of building a custom Python class from scratch. The goal of this class will be to solve a specific data-driven problem: handling geodata. We will show how to design the class to effectively manage this data, perform basic descriptive analysis, and display the results in a useful way. This will serve as a concrete example of how to create a reusable tool, directly preparing students for their final projects.

Lecture 12: Conclusion *F, A*

This lecture concludes the semester.

Exercise week 11-12: Q&A for the Project.

Tutorials

The course features 10 tutorial sessions designed to bridge the gap between lecture theory and practical application. These are supported by a team of teaching assistants.

Tutorial Structure

- First 8 tutorials: (Skill Building)
  - These are classic, hands-on sessions for students to practice the concepts from the lectures.
  - They will be held as a single session for all students, where one TA will lead the instruction while the other TA's will provide hands-on support and answer individual questions
- Final 2 tutorials: (Project Support)
  - These sessions are structured as "office hours" for the group projects.
  - Purpose: This time is dedicated for groups to meet with TAs to get help, troubleshoot problems, and receive direct guidance on their project.

TA Responsibilities & Budget

We request a budget of 80–90 hours per Teaching Assistant, divided in 5 TA positions.

The workload of TA's is relatively high for two reasons. First, because this is a new course: the teaching assistants will play a critical role in the co-development of course material for the tutorials. Unlike established courses where materials are reused, the TAs will be creating the tutorial exercises from scratch. Second, we need assistance of the TAs in grading the group projects. We have deliberately chosen a Group Project as the primary assessment method (70% of the grade). We believe this is superior to standard exams because it compels students to actively apply their knowledge, which is essential because programming is a skill mastered primarily through practice. However, this pedagogical choice creates a significantly higher workload than standard grading. TAs must perform detailed code reviews of Git repositories, verify the functionality of custom Python classes, and evaluate technical documentation. This requires TAs to act as technical mentors and reviewers, rather than simple markers.

Estimated workload breakdown (per TA)

- Grading of group projects and exams (30 hours)

TAs will assist in evaluating the final group projects. This involves reviewing the code repositories for quality and style, verifying the functionality of the Python classes, and assessing the documentation notebooks (Analysis / User Guide).

- Content creation and preparation (25 hours)

TAs will develop and refine the exercise materials and sample solutions for the 8 skill-building tutorials, ensuring they align with the weekly lecture content.

- Presence and teaching (15 hours)

In addition to the lead teaching role (rotating), TAs provide intensive in-class support, which is critical for programming courses with beginners/intermediates.

- Student support and admin (10 hours)

During the project phase, TAs act as technical mentors, helping groups structure their code and resolve Git/collaboration conflicts.