

# Schedule

## Contents

- Schedule
- Learning objectives
- The flipped classroom concept
- Assessment
- Literature
- Required programmes

Classes take place 9:30am-1pm via Zoom.

1. Monday, 23 Sep: **Introduction, first steps with the shell and git**
  - Basic shell usage
  - Mechanics of committing, pushing and pulling
2. Wednesday, 25 Sep: **Basic Python I**
  - Assigning variables
  - Important built in types
  - How to run .py files
  - How to read tracebacks
3. Friday, 27 Sep: **Basic Python II, Intermediate Git**
  - Avoiding code duplication with loops and functions
  - Conditional statements
  - What are libraries, importing
  - Git: undoing things, branches, merging + conflicts
  - Github and collaboration
4. Monday, 30 Sep: **Mechanics of pandas + pytask**
  - The pathlib module
  - pandas basics
  - pytask basics

[Skip to main content](#)

- Theory of data management (normal forms)
- More pandas
- More pytask
- Simple plots

#### 6. Monday, 7 Oct: **Project Templates**

- Definition of reproducibility
- Set up reproducible research projects based on [econ-project-templates](#)

#### 7. Wednesday, 9 Oct: **Software engineering**

- Styleguides and pre-commit
- Variable names, function names, ...
- Overview of the best-practice screencasts and when to watch them
- The importance of testing and specialised libraries for doing so

#### 8. Wednesday, 30 Oct: **Discussion of student projects**

Please set aside several hours of preparation time in between classes, see below why.

## Final project due on 27 October

- Between 9 October and 27 October, you will prepare a project (see below) and we will discuss that in the last session.
- Credit will be based on handing in a reasonable project

Modern didactics starts by defining a set of **active** knowledge you should have developed at the end of the course. The course contents are then derived from that and so is the [teaching style](#). Finally, the [things we require you to do in order to pass the course](#) are designed to require exactly those skills that we hope you will develop.

We group the learning objectives into two categories:

- Methods (core concepts and abstract coding skills)
- Tools (specific programmes, which you may want to use in the long run or not)

[Skip to main content](#)

# Methods

- Students master the basic techniques to make current and past versions of their research reproducible
- Students solve reasonably complex economic problems using the help of a computer
- Students are able to write correct code in an efficient manner employing modern software engineering techniques
- Students reason about the structure of economic data and handle such data efficiently

# Tools

- Students use a shell for navigating the directory system and for starting programmes
- Students routinely use *Git* for version control and collaborating with coauthors
- Students manage economic data using basic *pandas* functionality
- Students automate repetitive tasks using Python
- Students run their analysis pipelines using *pytask*
- Students find solutions to specific programming problems through sensible use of AI, search engines, and websites like *Stack Overflow*

*(probably not enough time for the rest)*

- Students visualize data using *plotly*
- Students speed up their code using *numpy* and *numba*
- Students write automated unit tests using the *pytest* framework

We (or rather generations of students) have found out that acquiring these skills through frontal lectures is not very efficient. Instead, the micro-schedule will be as follows:

- Before class, there will be:
  - Self-paced learning of concepts / knowledge. We will provide you with screencasts, literature, or short programming problems that will allow you to acquaint yourself **at your own pace** with the material.
  - Short quizzes, which allow you to check your knowledge. Note that these do not constitute tests. They do not enter the grading in any way, they

[Skip to main content](#)

- During class, there will be
  - Carefully designed group work in order to deepen your understanding of certain topics
  - Class discussions to collect the results from the group work and to foster presentation skills

Combined, these elements should allow you to

- acquire the basic knowledge at your own pace
- deepen it in group work

## Grading

As the teaching style, our choice of assessment criteria is guided by the learning objectives. In particular, there will be a final project of your choice, which typically would involve bringing a research project of yours into shape.

Overall, you should convince me that you have met the learning objectives of this course relating to “methods”. I.e., very abstractly, you are able to solve interesting economic problems that require the help of a computer in a way that is reproducible by others.

Here are some more detailed notes. Most probably, they are not comprehensible before the course but they will be helpful once you get started.

### 1. Please work on Github.

The last time we checked, GitHub placed a hard limit of 100MB on individual files. So do not include such files in your repository.

### 2. Grading will be based on programming style, task complexity, how much time it takes to understand where you are headed, whether your results are reproducible (depends a bit on the project), and your use of Git.

Making use of the project templates is by no means a requirement, but doing so will be useful for most projects. Webscraping is the clear exception here (i.e., this would be a separate step before applying the pytask machinery to the scraped data). In any case, it needs to be clear why the structure you chose for your project is useful for what you want to get done.

### 3. It is no requirement that we are able to run your code. However, if no data or

[Skip to main content](#)

But you should use whatever data or software are most useful for you.

4. In any case, provide a README that specifies the environment needed to run your project. That is, anything not included in your Github repository. This includes all required software (including things like Python libraries, R packages, Stata ado-commands, ...), on which operating system it has been tested, external data, etc.. If you just use Python and/or R, the most helpful strategy is to use a conda environment file.
5. **When you are done, place a tag called `epp-final-project` on the commit that we should look at. Add us (janosg, hmgaudecker) to the project.**

With the growth in the importance of data and various reproducibility crises (or just a constant state of crisis?), there has been a surge of efforts putting ideas similar to this course into writing.

Here are some with different focuses

- [Coding for Economists](#) Pretty cool, fairly recent online book, focus towards metrics.
- [Research Software Engineering with Python](#) — the closest in many ways to this course. One author is [Greg Wilson](#), who founded the [Software Carpentry](#) project. EPP grew out of “translating” an early version of Software Carpentry to economics in 2010. Naturally, we took some different branches since, but the core ideas remain the same.

At the time of this writing, some other book projects are underway by the same team, do check out <https://merely-useful.github.io/>.

- [The Plain Person’s Guide to Plain Text Social Science](#) — similar ideas, different angle. The introduction is a pure piece of beauty and gives some great background if you have mostly worked with Tablets, Word, and similar tools (as opposed to programming a lot yourself)
- [The Turing Way](#) — self-identifies as “an open source community-driven guide to reproducible, ethical, inclusive and collaborative data science.” Geared a bit towards larger projects, many very nice ideas.
- [Code and Data for the Social Sciences: A Practitioner’s Guide](#) — From 2014 and some specifics are a bit outdated. Also, we go well beyond this in the course. However, if you feel that you need some premier economists’ writing on the topic to gear up your motivation, this is for you.
- [A Gentle Introduction to Effective Computing in Quantitative Research](#) — A bit

[Skip to main content](#)

software engineering side. Very useful as a complement.

## 1. [Visual Studio Code](#)

The most important programme when coding is a versatile editor. VS Code is cross-platform, stable, free, fast, and very extensible. When installing on Windows, be sure to tick all boxes so you register it as the default editor.

Feel free to install the same packages as Hans-Martin did, which you can find [here](#).

Crucially, we will need the [Live Share extension](#), which will allow joint collaboration in groups.

## 2. Three more (sets of) programmes:

- The [Anaconda Python Distribution](#) including Python version 3.11
- [Git](#)
- A modern LaTeX distribution (e.g. [TeXLive] (<file:///home/janos/Downloads/www.tug.org/texlive/>), [MacTeX](#), or [MikTeX](#)).

You can just follow steps 1.-3. on [this page](#).

Previous  
< [Course Overview](#)

Next  
[To do before the first session](#) >