Lecture 1: Introduction to Programming
Programming for Economics—ECNM10106

Albert Rodriguez-Sala & Jacob Adenbaum
School of Economics, The University of Edinburgh

Winter 2023

This Class

Course Organization

Why Economists Program?

Learning Programming

Programming language: Python (and a bit of Julia)

Infrastructure

Course Organization

Syllabus

- Week 1: Introduction to programming.
- Week 2: Programming fundamentals.
- Week 3: Working with data.
- Week 4: From data to models.
- Week 5: Numerical Methods I.
- Week 6: Numerical Methods II.
- Week 7: Numerical Methods III.
- Week 8: Economic models.
- Week 9: Model Estimation.
- Week 10: Advanced Topics.

Some references and useful resources

- Numerical Methods in Economics, by Kenneth L. Judd, 1998. Main reference for weeks 4-8.
- Quantitative Economics founded by T. Sargent and J. Stachurski. Main reference.
- Lecture notes by J. Druedhal and C. Carstensen. Introduction to Programming and Numerical Analysis, University of Copenhagen.
- Advanced: Lecture notes by Jesús Fernandez-Villaverde. Computational Methods for Economists, University of Pennsylvania.
- For data analysis: Stata to Python equivalences by Daniel Sullivan.
- Python-Julia (and Matlab) cheatsheet.

Lectures, tutorials, and grading

- **Lectures:** cover the material. Always bring your laptop to lectures and tutorials. Mondays 13:10-15:00.
- **Lab/tutorials:** cover the problem sets (past-future). Might ask you to show/present your answers at some point. Starting on week 2. Tuesdays from 14:10 till 17:00 (3 groups). Thursday on-line catch-up tutorial 10:00-10:50.
- **Assessment**:
  - 8 group problem sets: 50%.
  - 1 individual take-home exam: 50%.
- **Office hours** (Albert): Thursdays 11:00 to 12:00. Location: Office 4.05, 30 Buccleuch place.

Problem Sets

- The problem sets should be completed in groups of 5 members.
- **Deadline:** Mondays 20:00.
- **Submission platform**: Each group must have a **Github repository** where the answers of each problem set are submitted.
- Submission of the problem set must consist of a SINGLE DOCUMENT containing all the answers and the code for all the exercises. The document must be called: PSX_groupY.
- **Accepted formats**: *.ipynb*, *.html*, *.pdf*, or *.doc* or any other file that we can read your answers. We require you to answer the questions in a text format and attach the code you used.
- If your problem set is not in a notebook, we ask you to UPLOAD THE SCRIPT (*.py,.jl*). We might check if your code runs and delivers the outcome you show us.
- Late assignment policy

Final Project

Individual take-home exam. You'll have a week or more to complete the exam. Late submission not possible.

Should expect a set of exercises that cover most of the material seen in class.

We will provide you more information about the final project as the course advances.

Why Economists Program?

Why economists program?

" *The era of closed-form solutions for their own sake should be over. Newer generations get similar intuitions from computer-generated examples than from functional expressions*",
Jose-Victor Rios-Rull (2008).

Why economists program? quantitative economics

- Economic models can be complicated $\rightarrow$ no closed-form (explicit) solution.
- More freedom of research $\rightarrow$ not subject to "simple" models with closed-form solutions and strong assumptions.
- Even if we can describe the qualitative results of theories, quantitative methods allow us to QUANTIFY THEORIES and QUANTIFY POLICIES.

Quantitative economics: Not only how but how much

- **Standard approach**: theory$\rightarrow$ test mechanism/policy in reduced form with the data.

- **Quantitative approach**: theory $\rightarrow$ quantify mechanism/policy in the data.
  1. How much mechanism can account for data trends and facts?
  2. How much policy intervention would affect society outcomes? Changes in GDP, inequality, welfare?

Programming used across all economic fields

- **Micro**: solve structural models for a better understanding of micro-mechanisms and a more completed policy evaluation. Also study of networks, dynamic games, etc.
- **Macro**: solution and estimation of dynamic equilibrium models, policy evaluation and forecast. In macro one cannot run experiments. Quantified models allow us to run counterfactuals on environment changes or policy changes (welfare analysis).
- **Applied**: Process large data sets with non-standard economic information: satellite pictures, weather data, historical data, text-mining, big-data, etc. Machine learning tools, non-standard estimators, simulation-based estimators, and other econometric tools.
- **Others Fields**: Dynamic models of international trade, spatial models, economic consequences of climate change and environmental policies, asset pricing models (finance), etc.

Wages are higher

Programming and quantitative economics skills are highly valued in the labor market

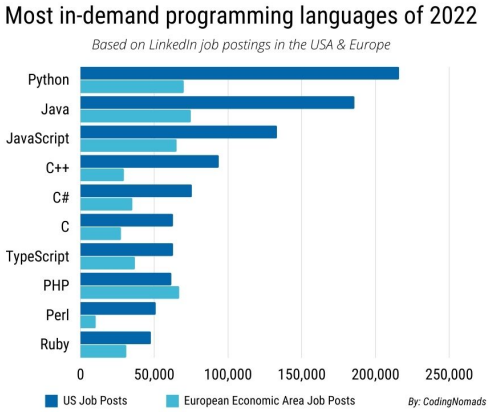


Figure 1: Linkedin job postings across programming languages. Source: Coding Nomads.

Some examples

- Macroeconomics and inequality.
  Castañeda, A., Diaz-Gimenez, J., Rios-Rull, J. V., (2003). Accounting for the US earnings and wealth inequality. Journal of Political Economy.
- Quantify welfare effects in macroeconomics
  Krueger, D., Mitman, K., Perri, F. (2016). On the distribution of the welfare losses of large recessions. National Bureau of Economic Research.
- Quantify the aggregate effects of policies (in development)
  Kaboski, J. P., Townsend, R. M. (2011). A structural evaluation of a large-scale quasi-experimental microfinance initiative. Econometrica.
- Quantify the welfare effects of policies (in education)
  Mullins, J. (2020). A structural meta-analysis of welfare reform experiments and their impacts on children. University of Minnesota, Minneapolis and Saint Paul, MN.

Learning Programming

The importance of theory in programming

This is not just a course on learning a program. Theory will be important.

- economic theory and mathematics will be very relevant for this course. Need to first understand the problem and theory to be able to write the code.
- We will focus on numerical analysis which requires us a well understanding of mathematics.

The three steps of numerical analysis

1. Mathematical model $\rightarrow$ create an algorithm
2. Algorithm [1] $\rightarrow$ write the code
3. Code $\rightarrow$ present your results

[1] Algorithm: A set of finite rules or instructions to be followed in calculations or other problem-solving operations. It can be understood as a cooking receipt but for a computational problem: ingredients (inputs), steps to execute one by one, new dish (output).

How do we program?

We never get right the code the first time. Is my problem in understanding the exercise? The mathematics behind the algorithm? Or in how to translate the algorithm into code?

1. Check lecture notes and the theory behind the exercise.
2. Write down in a paper what you want the code to do or the outcome to reproduce.
3. Sketch the code (pseudo-code) in a paper.
4. Bring the problem to the computer: write the code. More on this, next slides.
5. Present the outcome. Print function, plots, tables, etc

Writing code

- We all forget the correct syntax. Learning to write code is about learning how to properly use your past codes, code from others, and online resources. We never program from scratch.
  - **Google will be your best friend for this course.** We work with Python: Very popular open-software program. Great documentation with a huge community behind it. Julia is also open-software.
  - Stackoveflow: Almost all code doubts that you might have, are already asked (and answered) here.
- **Clear and understandable code.** Make use of comments to document what you do. Keep the code clean and tidy. Readability beats cleverness.
- **Comprehensive testing.** As you work on your code, test it. Use of print command, check the data type and format, use debugging tools.

Good coding practices

- Keep your code simple, tidy, and easy to read: Readability $>$ Cleverness.
- Break complex problems in small taskes: Divide and Conquer.
- Make frequent use of functions.
- Don't use magic numbers. No numbers scattered around: Assign them in a constant.
- Don't Repeat Yourself Principles.
- Be lazy, automate: First think and plan-ahead to avoid you overwriting.
- Organize your time efficiently. Breaks are necessary when coding.

Getting help

Steps to follow when you are stuck on the code

1. Look in lecture notes, our codes, your past codes, and online documentation.
2. Search: Google + Stackoverflow.
3. Talk about it in your group.
4. Help each other. Except on the final problem set.
5. We won't answer emails about code troubles like "why is my code not working?".

Programming language: Python (and a bit of Julia)

Python

- Python is a **general-purpose** programming language conceived in 1989 by Dutch programmer *Guido van Rossum*.
- **Free** and **open-source** language.
- Python has experienced rapid adoption in the last decades and now it is one of the most **popular** programming languages. See The Economist article.
- Used by the scientific community, CIA, Google, Spotify, Pixar, etc.

Figure 2: Guido van Rossum

Pros of Python

- Most **trending** coding language: great documentation with a huge community behind. Many developers, collaborators, and ample learning sources.
- **Versatile language**. Python is a general-purpose language used for: communications, web development, multimedia, **scientific computing, data science,** machine learning, etc.
- Free and open-source language.
  - Open-source: you can access all the primitive code from any routine, function, or package. And modify the code to your taste.
  - Nice IDEs available. Can set up your own **beautiful display**, not like Stata...
- Elegant syntax and elegant desing: much easier to learn than other programming languages (C, C++, Java, etc.).

Cons of Python

- Some of the pros of Python might be a con.
  - We might prefer non-general-purpose program (as Matlab, Stata) if our work is just centered in a specific "field".
  - Commercial languages (Matlab, Stata) offer better support than free open-source languages.
- **Execution time is** relatively **slow**. Alternatives: cloud computing, parallelization, other programs as Julia.

Pros and Cons of Julia

Pros

- Julia shares many of the pros in Python: free and open-source language, beautiful display, elegant syntax, quite versatile, and it is getting trendy in academia.
- **Just-In-Time compliation**: code execution much faster than in Python. This is specially important when working with large-scale problems.

Cons

- Not as popular as Python (yet): Less resources, less documentation. Little known outside academia.

Python features

- Python is a high-level language suitable for rapid development. It has a relatively **small core language** supported by **many libraries**. Main libraries we will use are
  - NumPy: fundamental library for scientific computing.
  - PanDas: fundamental library for data manipulation and data analysis.
  - SciPy: fundamental library for numerical analysis: optimization, integration, interpolation, etc. Written in low-languages (Fortran, C, and C+) enjoying the speed of compiled code.
  - Matplotlib: main library for plots and visual representation.
- Python supports **multiple programming styles** (procedural, object-oriented, and functional. Yet, it is mainly design for Object-Oriented Programming.
- it is **interpreted** rather than compiled. Execution time is low.

Python is based on Object Oriented Programming

Python is based on OOP but also supports procedural, and functional programming. Matlab, Stata are procedural-based. Julia also has multiple programming styles but is better suited for a procedural approach.

- **The procedural paradigm:** Program has a state that contain values of its variables. Functions are called to act on these data, data are passed back via function calls.
- **The OOP paradigm: data** and **functions** are **bundled together** into **objects**
  - In the OOP functions are usually called methods.
  - In Python the data and methods of an object are referred to as attributes. Depending on the class of object (like float, array, string, dataframe, etc) the object will have different methods.

Example: Procedural vs OOP

Since Python supports both procedural and OOP we can use both approaches. Note that many functions will also be a method and viceversa.

Example

Compute the mean of a matrix $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

```
# Import NumPy library to work with matrices/arrays.
import numpy as np
A = np.array([[0,1,0],[0,1,1]])  #create the 2-D array---i.e. matrix
# Procedural approach. We call np.mean function to act on A data.
 np.mean(A)
 0.5
# OOP approach. We call the array-method mean to act on the array-object A.
 A.mean()
 0.5
```

Object Oriented Programming

Object Oriented Programming OOP is useful as for the same reason abstraction is: for recognizing and exploiting common structure.

In Python everything in memory is treated as an object. This includes not just lists, strings, numbers, but also datasets, functions, etc.

An object is
a collection of data and instructions held in computer memory that consists of:

- a type,
- some content,
- a unique identity,
- methods.

Is Julia OOP?

Julia is a multi-paradigm language and can include OOP. Yet, most problems in Julia are better suited to a functional language than OOP. Discussion

Infrastructure

Infrastrucutre

Python and Julia, are the programming languages. Now we need to download the "programs"—the Integrated Development Environments (IDE)—to use the languages.

- for Python:
  - The **Anaconda** distribution.
  - IDE (code editor): **Spyder**.
  - Notebooks in **Jupyter Notebooks**.
- **IDE in Julia: VS Code**. Installation guide.
- We will use **Github** to commmunicate and share codes and material. Including your problem sets submissions.

Anaconda

Installing **Anaconda**:

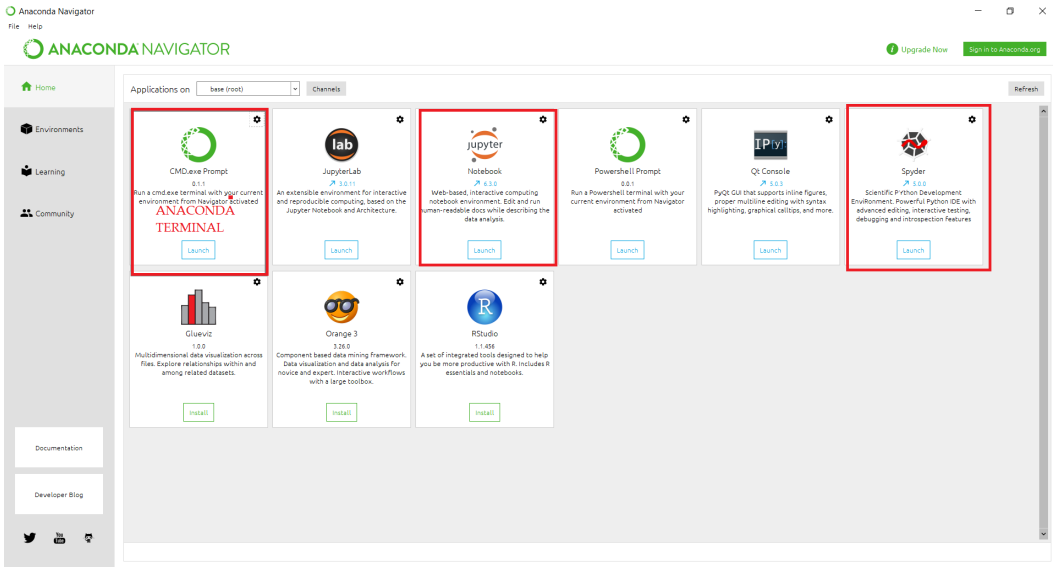
1. Download Anaconda Individual Edition Python 3.9 from Anaconda website.
2. Run the installer (default settings are fine)

With Anaconda we already download our main Python IDE, **Spyder**, the useful **Jupyter Notebooks**, and the IDE+notebooks in Julia **VS editor**.

Anaconda's terminal *CMD Prompt*.

- To open a program type the name of the program in the terminal: `spyder` `jupyter notebook`
- To update a program: `conda update spyder`
- To install a package—Example: installing the quantecon package. `conda install -c conda-forge quantecon`

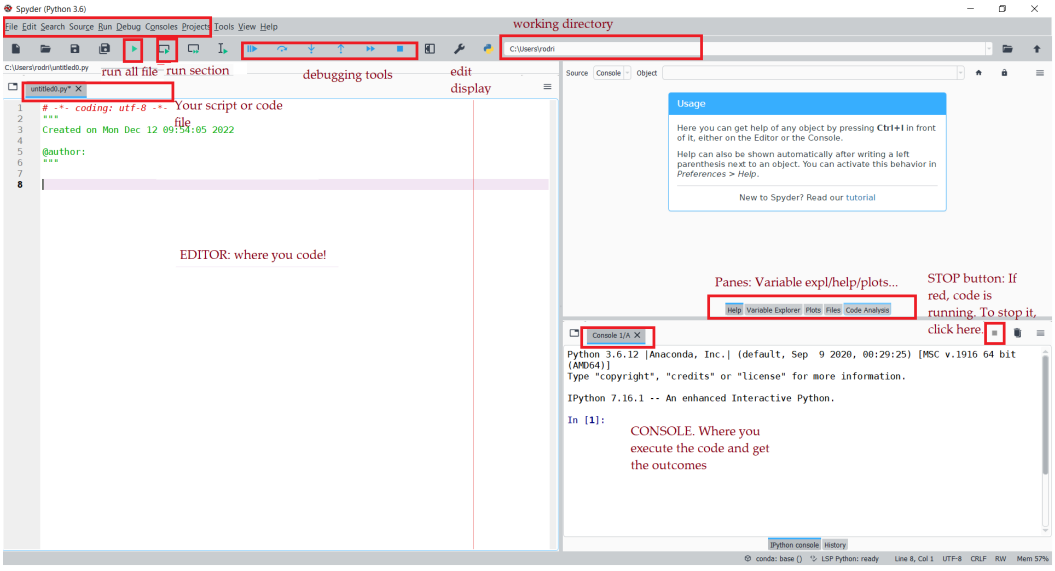
Anaconda desktop



Spyder

- Integrated Development Environment (IDE) for Python.
- *Spyder is a free and open source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts.*
- Quick-start tutorial.
- Intro Videos.

Spyder desktop



Spyder desktop

Toolbar: quickly access some of the most common commands in Spyder, such as run, save and debug files.

Panes:

- **Editor:** Create, open, and edit script files—where you code. Editor tutorial for a detail explanation, key components, shortcuts and personalization features.
- **I-python Console:** Exectue commands and interact with data inside IPython interpreters—where the code is executed. Tutorial
- **Variable explorer:** allows you to interactively browse and manage the objects generated running your code. Where your outcomes get stored.
- **Help:** Display documentation for the objects you are using in the Editor or the IPython console.
- **Other panes**: Plots/History/Files/Code analysis.

Spyder does not work? Troubles? check Spyder basic first aid.

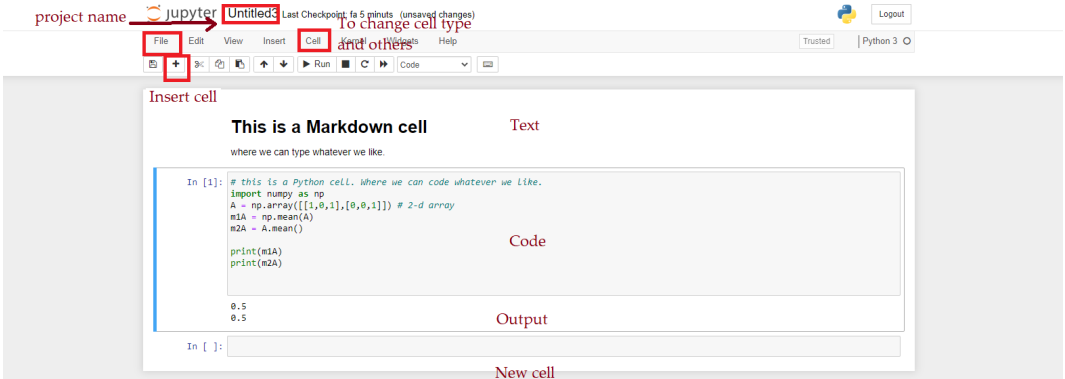
Spyder shortcuts

- F9: run selected lines.
- Ctrl+enter: Execute cell.
- Ctrl+F: Search
- Ctrl+R: Search and replace.
- Ctrl+S: Save.
- Ctrl+T: New console.
- Ctrl+I: Help
- Ctrl+Z: Undo.
- Arrow-up: In console to get the code previously executed.

Jupyter Notebooks

- Notebooks are very useful to present your code in results. We recommend you to work on the problem sets on Spyder and then submit the code+answers in a Jupyter notebook (nice and efficient).
- Tutorial
- Starting Jupyter notebook:
  1. Open Anaconda Navigator and launch Jupyter Notebook by mouse click. Using terminal: Open anacond prompt (windows) or Terminal (Mac). Write Jupyter notebook and hit enter.
  2. - Once launched: select your working directory folder. Click on *New*, click on *Python 3*. You have just created an empty Jupyter notebook!

Jupyter Notebooks desktop



Notebooks

Notebooks consists of two types of cells:

- Code cells with Python code
- Markdown cells with text. Short markdown tutorial

When inside a cell you are in edit mode, when not you are in command mode. To change the cell type your are, go to *Cell* – C*ell type* and select *code* or *Markdown*.

Jupyter notebook Short-cuts

- Run cell: Ctrl+Enter.
- Enter edit mode: Enter.
- Enter command mode: Ctrl+M

Jupyter notebooks

Notebooks are the nicest and more efficient way to present your exercises. We expect you to submit the homework in notebooks.

- Save and output format of notebooks: You can save your current notebook (*.ipynb*) using save as or the download option. You can submit your problem set as a *.ipynb*.
- You can also directly download your notebook to a pdf. File-download as–pdf. For that, though you need to first install a plug-in. You can also download the notebook in *.html* or another type of file.
- Files in *.ipynb*, *.html* can be directly opened in Github (and in general in browsers). We recommend you to submit your homeworks in these formats, specially *.ipynb*.

Git and Github

- **Git** is a version-control system that virtually everyone in CS and scientific programming uses. It allows programmers to coordinate their work across computers without messing up.
- **GitHub** is the main online platform for sharing work through Git. Thus, in this work we will share our work and your work) in Github.

Working with Git and Github and Anaconda (download modules uploaded in github).

- Install the git package in Anaconda: `conda install git`
- To install a library in a Github repository use *git clone +github-repository-link*.
  Example: `git clone https://github.com/QuantEcon/QuantEcon.py`

We ask

- All of you to open an account on Github.
- Each group to create a Github repository to submit the problem sets.

Github repository to submit your problem sets

- **Create a single repository** for your group. The repository must be **private**. Steps here.
- **Invite collaborators**: you should invite your group members, Jacob (username: jacobadenbaum) and Albert (username:albertrodriguezsala). Steps here
- In this repository you'll upload your problem sets and code.
- To add a file: go to *new file-add file*. Choose the file and commit changes.
- Submission must consist of a SINGLE DOCUMENT containing all the answers and the code for all the exercises. The document must be called: *groupX_psY* (where X is your group number, Y the problem set number).
- Accepted formats: *.ipynb*, *.html*, *.pdf*, or *.doc*. The file must contain the answers and the code.
- If your problem set is not in a notebook, you should also upload the script(s) (.py, .jl).

Your To-Dos

- Download Anaconda. Open Spyder and Jupyter Notebooks. Check and run *code_example_1.py*.
- Form your group for the problem sets (5 members) and register in the Econ-10106 groups Google Sheet.
- Create a Github account. Create a unique Github repository for your group.
- Each group send me (Albert.Rodriguez@ed.ac.uk) one email with the link to your group repository. Make clear your group number and members of the group in the email.
- Work on the problem set 0 and upload it in your group's repository (whatever you managed to do).