

Lecture 1: Introduction to Programming Programming and Numerical Methods for Economics—ECNM10115

Juan Zurita & Zhuotong Liu

School of Economics, The University of Edinburgh

Autumn 2025

This Class

Course Organization

Why Economists Program?

Survey

Learning Programming

Programming language: Python

Infrastructure

Course Organization

- Week 1: Introduction to programming.
- Week 2: Fundamentals of programming in Python.
- Week 3: Data manipulation and data analysis.
- 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.
- Advanced: [Lecture notes by Jesús Fernández-Villaverde](#). Computational Methods for Economists, University of Pennsylvania.
- For data analysis: [Stata to Python equivalences](#) by Daniel Sullivan.
- [Python-Julia \(and Matlab\) cheatsheet](#).

- **Lectures:** cover the material. Always bring your laptop to lectures and tutorials. Thursdays 15:10-17:00.
- **Lab/tutorials:** cover the problem sets (past-future). Might ask you to show/present your answers at some point. Starting on week 2.
- **Assessment:**
 - 8 group problem sets: 40%.
 - 1 individual take-home exam: 60%.
- **Office hours** (Juan): Fridays 16:00 to 17:00. Location: Office 2.11, School of Economics.

Problem Sets

- The problem sets should be completed in groups of 3 members.
- **Deadline:** Tuesday 2:00 pm.
- **Submission platform:** Each group must submit the answers of each problem set using Learn. It is a group submission.
- 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*, *.pdf*.
- If your problem set is not in a notebook, we ask you to UPLOAD THE SCRIPT (*.py*). We might check if your code runs and delivers the outcome you show us.
- Late assignment policy
- **Marking Criteria available in the course outline on Learn**

Table 1: Problem Set Grades Spring 2025

Statistics (All Groups)	PS1	PS2	PS3	PS4
Mean	65	72	70.2	74.2
Max	73	79	72	78.4
Min	50	63	67	69.76
Sd	8.2	5.1	1.8	3.1

Individual take-home exam of 2-3 days.

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.

Figure 1: Statistics

Statistic	Value
Minimum	34.50
Median	70.00
Maximum	83.50
Mean	68.16
Std	10.31
Number of Exams	46

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 → test mechanism/policy in reduced form with the data.
- **Quantitative approach:** theory → quantify mechanism/policy on 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, etc.

Example: Programming in Macroeconomics

Household maximisation problem, time-separable life time utility subject to her budget constraint:

$$\max_{\{c_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t E \underbrace{[\ln(C_t)]}_{\text{Utility Function}} \quad (1)$$

$$\text{s.t.} \quad \underbrace{K_{t+1} + C_t = Y_t + (1 - \delta)K_t}_{\text{Budget Constraint}} \quad (2)$$

where now

$$Y_t = z_t K_t^\alpha \quad (3)$$

$$\log(z_t) = \rho \log(z_{t-1}) + \sigma \epsilon_t \quad (4)$$

$$\epsilon_t \sim N(0, 1). \quad (5)$$

where C , K , Y denote consumption, capital and output. δ and z identify the capital depreciation rate and the aggregate shock.

Wages are higher

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

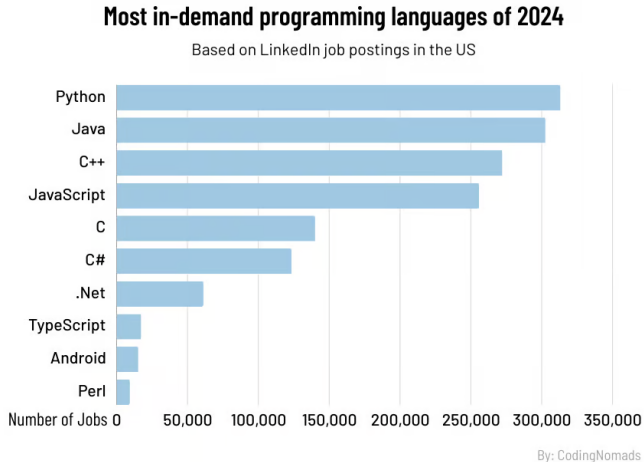


Figure 2: LinkedIn job postings across programming languages. Source: [Coding Nomads](#).

Some examples: Quantitative Economics

- 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.

Survey

www.menti.com

Password: 1908 4847



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 → create an algorithm
2. Algorithm ¹ → write the code
3. Code → present your results

¹Algorithm: 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

- 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.
 - [Stackoverflow](#): 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.



Figure 3: Coding is fun!

Good coding practices

- Keep your code simple, tidy, and easy to read: Readability > Cleverness.
- Break complex problems in small tasks: 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.

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

- 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 4: 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.).

- 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.

- 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 an array $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

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 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.

Infrastructure

Python is a programming language. Now we need to download the "programs"—the **Integrated Development Environments (IDE)**—to use the language.

- for Python:
 - The **Anaconda** distribution.
 - IDE (code editor): **Visual Studio Code**, **Spyder**, etc.
 - Notebooks in **Jupyter Notebooks** or [Google Colab](#).
 - Anaconda Installation [Installation guide](#)

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 **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

Anaconda Navigator

File Help

ANACONDA NAVIGATOR

Upgrade Now Sign in to Anaconda.org

Home Environments Learning Community

Applications on base (root) Channels Refresh



CMD.exe Prompt
0.1.1

Run a cmd.exe terminal with your current environment from Navigator activated

**ANACONDA
TERMINAL**

Launch



JupyterLab
3.0.11

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launch



Notebook
6.3.0

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch



Powershell Prompt
0.0.1

Run a Powershell terminal with your current environment from Navigator activated

Launch



Qt Console
5.0.3

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Launch



Spyder
5.0.0

Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Launch



Glueviz
1.0.0

Multidimensional data visualization across files. Explore relationships within and among related datasets.

Install



Orange 3
3.26.0

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

Install



RStudio
1.1.456

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Install

Documentation Developer Blog

Twitter YouTube GitHub

- 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

The image shows the Spyder Python IDE interface with several components labeled:

- run all file**: Points to the green play button in the toolbar.
- run section**: Points to the green play button with a magnifying glass in the toolbar.
- debugging tools**: Points to the toolbar area containing buttons for stepping through code (step over, step into, step out, etc.).
- edit display**: Points to the top-right pane, which currently shows a "Usage" help window.
- working directory**: Points to the text box at the top right showing the current directory path.
- EDITOR: where you code!**: Points to the main code editor area on the left.
- Panes: Variable expl/help/plots...**: Points to the tabs in the bottom-right pane.
- STOP button: If red, code is running. To stop it, click here**: Points to the red square stop button in the bottom-right pane.
- Console 1/A X**: Points to the console window tab in the bottom-right pane.

The code editor contains the following text:

```
1 #-*- coding: utf-8 -*- Your script or code
2 """
3 Created on Mon Dec 12 09:54:05 2022
4 file
5 @author:
6 """
7
8
```

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

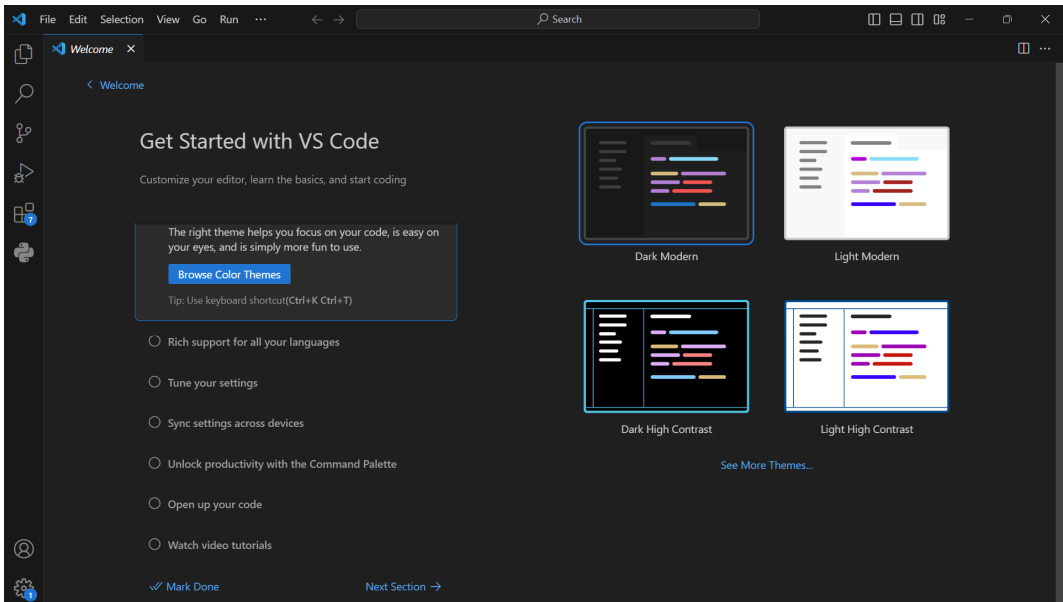
Panels.

- **Editor:** create, open, and edit script files—where you code. [Editor tutorial](#) for a detail explanation, key components, shortcuts and personalizing features.
- **I-python Console:** execute 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](#).

- 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.

Visual Studio Code



- 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

The screenshot shows the Jupyter Notebook desktop interface. At the top, the 'project name' is 'jupyter' and the notebook is titled 'Untitled3'. The last checkpoint was 'fa 5 minutes (unsaved changes)'. The top bar includes a 'Logout' button and a 'Python 3' environment selector. The menu bar contains 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar includes icons for saving, opening, and running cells. The main area displays a notebook with a 'Text' cell containing the text 'This is a Markdown cell' and 'where we can type whatever we like.' Below it is a 'Code' cell containing Python code for importing numpy, creating a 2D array, and calculating means. The output of the code is shown below the code cell. At the bottom, there is a 'New cell' button.

project name jupyter Untitled3 Last Checkpoint fa 5 minutes (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Insert cell

This is a Markdown cell

where we can type whatever we like.

```
In [1]: # this is a Python cell. where we can code whatever we like.
import numpy as np
A = np.array([[1,0,1],[0,0,1]]) # 2-d array
m1A = np.mean(A)
m2A = A.mean()

print(m1A)
print(m2A)
```

0.5
0.5

New cell

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 – Cell type* and select *code* or *Markdown*.

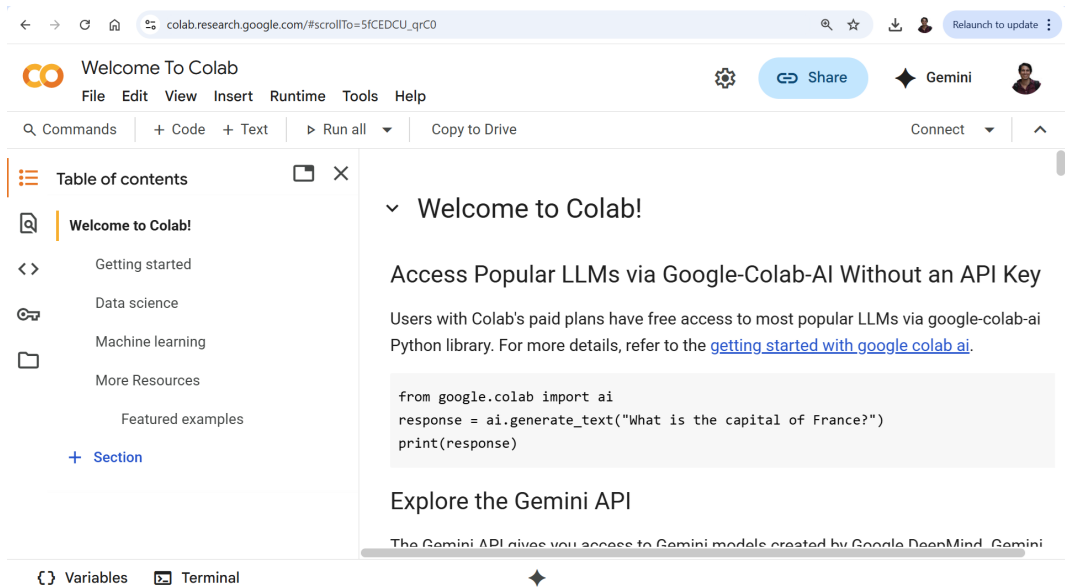
Jupyter notebooks short-cuts

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

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, you need to first install a **Pandoc** and **Miktex** through CMD Prompt.
- Files in *ipynb*, can be directly opened in Github (and in general in browsers). *.ipynb*. You can also print the notebook and presented and submit it in *.pdf*. We recommend you to submit your homeworks in these formats.

Google Colab: allows you to write and execute Python in your browser



The screenshot shows the Google Colab web interface. At the top, the browser address bar displays `colab.research.google.com/#scrollTo=5fCEDCU_qrC0`. The main header includes the Colab logo, the text "Welcome To Colab", and navigation links: File, Edit, View, Insert, Runtime, Tools, and Help. On the right side of the header, there are icons for settings, a "Share" button, the Gemini logo, and a user profile picture. Below the header, a secondary navigation bar contains "Commands", "+ Code", "+ Text", "Run all" (with a dropdown arrow), "Copy to Drive", "Connect" (with a dropdown arrow), and an upward arrow icon.

On the left side, a "Table of contents" sidebar is open, listing the following items:

- Table of contents (with a close icon)
- Welcome to Colab! (selected)
- Getting started
- Data science
- Machine learning
- More Resources
- Featured examples
- + Section

The main content area displays the "Welcome to Colab!" section. It includes the heading "Access Popular LLMs via Google-Colab-AI Without an API Key" and a paragraph stating: "Users with Colab's paid plans have free access to most popular LLMs via google-colab-ai Python library. For more details, refer to the [getting started with google colab ai](#)." Below this text is a code block containing the following Python code:

```
from google.colab import ai
response = ai.generate_text("What is the capital of France?")
print(response)
```

Below the code block, the section "Explore the Gemini API" is visible, followed by the text: "The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini".

At the bottom of the interface, there are icons for "Variables" and "Terminal", and a central diamond-shaped icon.

Your To-Dos

- Download Anaconda. Open Jupyter Notebooks, VSCode or Spider. Check and run *code_example_1.py*.
- Form your group for the problem sets (3 members) and register in the [Econ-10115 - PNME - PS Groups](#)
- Work on the problem set 0 and upload it in your group's repository (whatever you manage to do).
- Use the Python Refresher if you need.