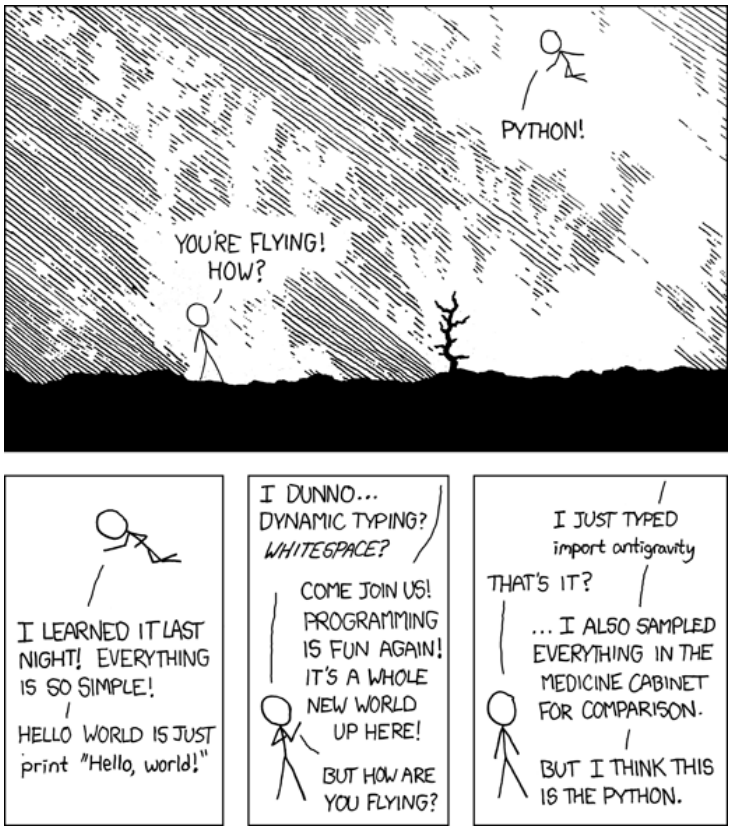
POL42340
Programming for Social Scientists

Johan A. Dornschneider-Elkink

School of Politics and International Relations
University College Dublin

http://www.joselkink.net

Spring 2025



Introduction

This module provides an introduction to computer programming using the Python language. Python is the most popular programming language at the moment,[1] including among data scientists, and is generally known as an excellent language to learn programming. A basic grounding in programming will allow you to automate mundane and repetitive tasks, for example renaming files, extracting web data, or developing exciting social science simulations. These are all applications that are typical for a social scientist. Or you can just use the knowledge to develop fun games as apps on your phone.

You do not need to have any prior knowledge of Python or computer programming to take part in this module. Our main project will be a social simulation, which will be developed in teams. While all students will learn the basic programming skills, groups of students will be assigned different aspects of the program, while sharing their experience with the rest of the class. This will allow us to cover a wide range of aspects of the program (configuration files, output files, visualisations, user interface, etc.), while keeping the overall effort manageable for you.

Rather than using Python as a typical scripting language, with a short set of simple commands to do some basic tasks, we will focus on the design of somewhat larger software projects. You can think of larger social simulations or a more complicated computer programme. The course is designed to teach you the very basics of *object-oriented software design*. For those students already familiar with some basic script writing in R or other statistical software package, this will be a very different programming experience. Understanding these general approaches to programme design will help you in all your programming tasks, small to large.

By the end of the module, students should have:

- Understanding of the relevance of computer programming in the social sciences
- Foundational level knowledge of the Python programming language
- Solid experience in team-based development
- Basic experience with collaborative programming tools
- Good grasp of key concepts in object-oriented programming
- Basic understanding of developing social simulations

Classes

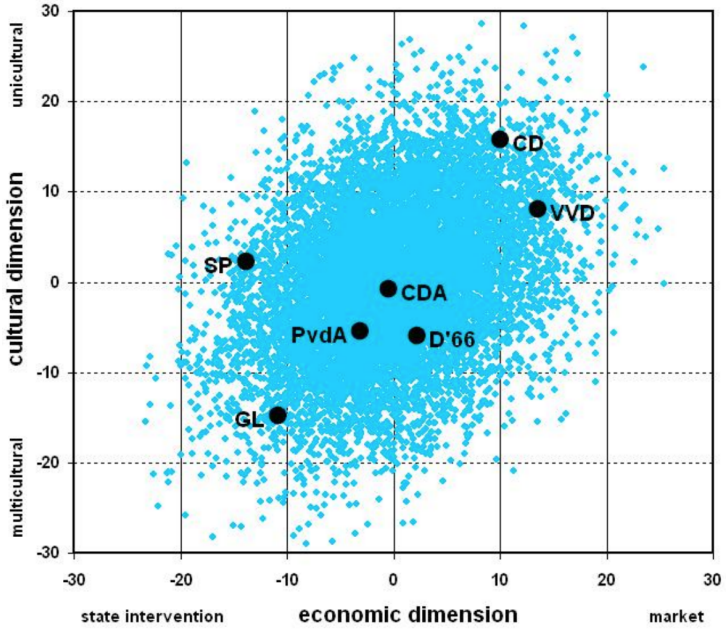
Classes take place once a week, from 14–16 on Fridays, in Room B110 in the Newman Building (except the 28 March session, which will be in B109). Classes will consist of a combination of lectures, group discussions, and programming labs.

[1] `https://www.tiobe.com/tiobe-index/`, accessed 16 January 2025.

Project

An agent-based model is a (computer) simulation where group-level behaviour is investigated based on individual-level behaviour. In an agent-based model we are typically trying to understand how a combination of different types of agents in a large system interact, and how this leads to perhaps surprising outcomes at the aggregate, social level.

When we study political parties, we are often interested in how political ideology affects voters' and parties' behaviour. We typically assume that voters will vote for the party ideologically closest to them. What kind of party behaviour would that encourage? Is it clever for the party to be ideologically in the centre, to attract a wide range of voters? Or is it smarter to find the middle ground among your current voters and solidify your position that way? Laver (2005) proposes a computer simulation to investigate different party strategies in an ideological space. This is an example of an agent-based model.



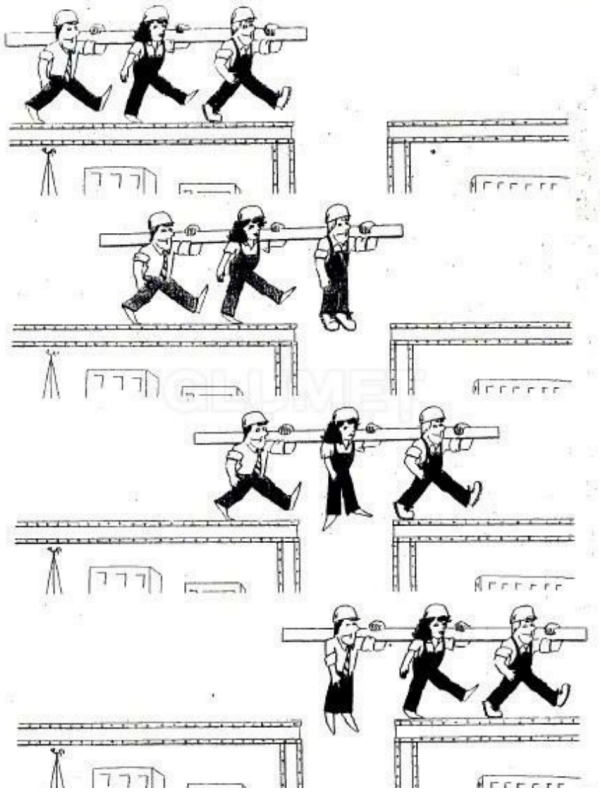
An agent-based model is a computer simulation where group-level behaviour is investigated based on individual-level behaviour. For example, if we assume that voters always vote for the nearest party (individual-level rule), and parties search aggressively to maximise their vote (another individual-level rule), how many parties will survive in a political system (a macro-level outcome)? In this course, we will replicate and possibly expand on the simulation by Laver (2005).[2] The simulation is relatively straightforward and easy to understand conceptually, but also leaves extensive scope for additional features. It is therefore ideal for a basic introduction to computer programming and simulation.

Programming skills are useful for a range of applications in the social sciences, including statistical data management, transforming unstructured data into useful statistical data, scraping

[2]Examples of extensions include adding more ideological dimensions, death and birth of parties, different electoral voting procedures, etcetera. Or adding media as in Muis (2010), which is the basis of the figure shown above.

data from the web, developing computer programs for lab experiments, general manipulation of files and automation of tasks, statistical analysis, and computer simulations. In the project, we will implement a well-known simulation, with features such as making it easy for the user to change settings, or saving the outcome of the simulations in an easy file format. This way, many key features of programming will be encountered.

All students will work on the same project, working on specific features in small teams. Imagine you are hired by a software development company and you are just one member of the development team, working on this project. This is a realistic setup for professional life as a data scientist. We will make use of professional tools for collaboration in programming, which will be helpful in your future career.



Teamwork will save you from falling!

Class seminars will be used to discuss overall progress, obstacles encountered, ideas for next steps or extensions, using the relevant tools, and so forth, while short lectures—and potentially some video lectures—will be used to teach the basic programming skills required.

You will not be assessed on the quality of the end product, but you will be assessed on your contribution and your ability to critically reflect on your participation in the project. In order to do so, you will submit a final progress report, where you focus on your contribution to the project, obstacles you encountered and how you addressed these, thoughts on how to improve the way the team operates or how you can improve your contribution to the project.

Assessment

The assessment for this module consists of a number of different components. For the above mentioned project you will submit a reflective report on the collaboration towards the end of the course. To test more basic programming skills, there will be three short MCQ tests in class on Python. Finally, there will be an exercise to design an object-oriented class diagram, which will be due later in the course.[3] This leads to the following distribution of grade components:

MCQ test 1	21 Feb	20%
MCQ test 2	28 Mar	20%
MCQ test 3	25 Apr	20%
Class diagram	28 Mar, 1 pm	10%
Lab report	8 May, 5 pm	30%

For late submissions the standard policies apply.[4] It should also be taken into consideration that a late submission might result in a delayed return of feedback to the entire class. Exemptions will be granted only on the basis of illness or bereavement, documented in all cases.[5]

Lab reports and class diagrams will be submitted through Brightspace. MCQ tests will take place on paper at the start of the class. MCQ tests will be marked using the "Alternative Linear Conversion Grade Scale 40% Pass" scale.[6]

Lab report

The purpose of the lab report is a self-reflection on your progress in programming and your progress in participating in the team project. While it should be formatted similar to other essays, the structure and contents can be quite flexible, depending on where you would like to focus. You can ask yourself questions such as the following to guide your writing—but these are just suggestions and do not need to be exhaustively discussed:

- What is my objective taking this course? How am I progressing on this? E.g. put this in context of your potential future career trajectory.
- What are difficulties I'm encountering in terms of the programming?
- What has been my role in the project thus far? Does this align with my expectations?
- Have I encountered difficulties interacting with classmates? Are particular behaviours of others difficult to deal with? How did I deal with this and how could I?

[3] For the diagram you can use Yuml, at `https://yuml.me/diagram/scruffy/class/draw`.

[4] `http://www.ucd.ie/governance/resources/policypage-latesubmissionofcoursework/`.

[5] `https://www.ucd.ie/science/study/currentundergraduatesciencestudents/extenuatingcircumstances/`.

[6] `https://www.ucd.ie/students/exams/gradingandremediation/understandinggrades/`.

- How is the social simulation project evolving? Do I feel I have a handle on what the simulation is trying to do? What does this project tell me about the possibilities for social simulation?

Throughout, try to focus on reflecting on your own expectations, activities, obstacles, and opportunities. Whether it relates to programming, social simulation, or team work, think about what your expectations are, what you did, what you found difficult, and what you think you can work on in future.

Reports that focus on one or two key issues and delve a bit deeper to evaluate what happened, what your actions were, how you could have addressed this differently, what you might have learned for the future, etcetera, are better than reports that address many issues briefly.

Do not use the above set of bullet points are section headers in the report, but come up with your own structure. But do use headings for different sections.

The lab report should be between 2,000 and 2,500 words.

Where your reflections involve interaction with other students, these reflections must remain anonymous! Do not name fellow students in your submission.

Plagiarism

Although this should be obvious, plagiarism—copying someone else's text without acknowledgement or beyond "fair use" quantities, or that of your own in another submission or publication—is not allowed. UCD policies concerning plagiarism can be found online.[7]

Contact

The main point of contact will be the Friday sessions and continuous interaction through our Matrix Space.[8] You can also try to find me in my office, room G309 in the Newman Building. The most reliable way to reach me is by email (`jos.elkink@ucd.ie`).

To stay up to date with developments in the UCD School of Politics and International Relations:
Web: `http://www.ucd.ie/politics/`
Blog: `http://politicalscience.ie/`
Twitter: `http://twitter.com/ucdpolitics`

[7] http://www.ucd.ie/governance/resources/policypage-plagiarismpolicy/

[8] A link to the Matrix Space will be sent separately. Many apps can be used to access this, but the easiest might be the default, Element: `https://element.io/download`.

Mastodon: `https://sciences.social/@ucdpolitics`
Facebook: `http://www.facebook.com/ucdspire`

Schedule details

24 Jan	**Introduction: Programming and simulations** Introduction to programming and Python. Working with code repositories. Introduction to the main project and social simulation. Lab: Experimenting with collaborative editing of code. Required reading: Lubanovic (2020, ch 1); Laver (2005) Further reading: Deitel and Deitel (2022, ch 1)
31 Jan	**Variables and functions** How to store data in the computer program. How to do basic calculations in Python. How to use functions to encapsulate programming steps. Lab: Implementing some functions of relevance to the simulation. Required reading: Lubanovic (2020, ch 2, 3, 9 up to p. 147) Further reading: Deitel and Deitel (2022, ch 2, 4)
7 Feb	**Conditions and flow** Basic introduction to boolean algebra and controlling the flow of the program. Lab: Implementing key data objects of the simulation. Required reading: Lubanovic (2020, ch 4, pp. 214–215) Further reading: Deitel and Deitel (2022, ch 3)
14 Feb	**Loops and lists** Python collections such as sets, lists, and dictionaries. Learning to work with loops and iterators. Lab: Implement the basic functionality of the simulation. Required reading: Lubanovic (2020, ch 6–8) Further reading: Deitel and Deitel (2022, ch 5–6)
21 Feb	**Working with text** How to input and output data to the user. How to manipulate text strings. Lab: Have the user input parameters of the simulation. Required reading: Lubanovic (2020, ch 5) Optional reading: Lubanovic (2020, ch 12) Further reading: Deitel and Deitel (2022, ch 8)
	In-class MCQ 1
28 Feb	**Types, classes, and object-oriented programming (1)** Introduction to the basic idea of object-oriented programming (OOP). Lab: Discussion of refactoring code to introduce object-oriented design. Required reading: Lubanovic (2020, ch 10) Further reading: Deitel and Deitel (2022, ch 10)
7 Mar	**Types, classes, and object-oriented programming (2)** How to structure code in a larger project in OOP. Lab: Implementing a refactored version of the simulation, using object-orientation and multiple files. Required reading: Lubanovic (2020, ch 11)

... table continues on the next page ...

28 Mar	**Reading and writing text files** How to input and output data to your program using files, including configuration and log files. Lab: Implementing a configuration input file for the simulation. Required reading: Lubanovic (2020, ch 14) Further reading: Deitel and Deitel (2022, ch 9)
	In-class MCQ 2
	Class diagram due
4 Apr	**Databases and data files** Basic discussion of databases, comma-separated files, and binary files. Lab: Implementing a data output file for the simulation. Required reading: Lubanovic (2020, ch 13, 16)
11 Apr	**Testing, debugging, and exception handling** How do deal with bugs and errors? Good practice in software development. Introduction to ideas of agile development. Lab: Adding logging to the simulation application. Required reading: Lubanovic (2020, ch 19) Further reading: Deitel and Deitel (2022, ch 9)
25 Apr	**Class inheritance** Basic theoretical introduction to class inheritance and design patterns in OOP. No lab Required reading: TBC Further reading: Gamma (1995); Giridhar (2016)
	In-class MCQ 3
8 May	*Final project report due*

References

Deitel, Paul and Harvey Deitel. 2022. *Intro to Python for computer science and data science*. Global edition ed. Harlow, UK: Pearson.

Gamma, Erich. 1995. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Giridhar, Chetan. 2016. *Learning Python design patterns*. Packt Publishing Ltd.

Laver, Michael. 2005. "Policy and the dynamics of political competition." *American Political Science Review* pp. 263–281.

Lubanovic, Bill. 2020. *Introducing Python: Modern computing in simple packages*. 2nd edition ed. O'Reilly.

Muis, Jasper. 2010. "Simulating political stability and change in the Netherlands (1998-2002): An agent-based model of party competition with media effects empirically tested." *Journal of Artificial Societies and Social Simulation* 13(2):4.