PRINCIPLES OF PROGRAMMING

Grado en Computación e Inteligencia Artificial / Bachelor in Computer Science and Artificial Intelligence null SEP-2025
POP-CSAI.1.M.A

Area Computer Science
Number of sessions: 30
Academic year: 25-26
Degree course: FIRST
Number of credits: 6.0
Semester: 2º
Category: BASIC
Language: English

Professor: **NACHO MOLINA CLEMENTE**

E-mail: nmolinac@faculty.ie.edu

Nacho Molina is Professor of Machine Learning and Data Science in Biotechnology at IE University, and Head of the Health & MedTech Lab within the IEX Research Xcelerator. His research focuses on developing probabilistic and biophysics-informed machine learning models, including biophysics-informed deep learning, Bayesian inference, and nonlinear dimensionality reduction, to understand complex biological systems such as gene regulation, cell proliferation, and differentiation. He has designed computational frameworks like DeepCycle, FateCompass, FourierCycle, and HiddenFoot, which integrate variational autoencoders, kernel methods, and Gaussian processes with mechanistic modeling to analyze single-cell, spatial, and single-molecule omics data. His work bridges machine learning with systems biology and has been published in leading journals such as Nature Communications, PNAS, and Science.

Prior to IE, Prof. Molina served as a CNRS Research Director at IGBMC (Strasbourg), where he led an interdisciplinary group at the intersection of machine learning and regulatory genomics. He was Adjunct Professor at the University of Strasbourg, where he taught machine learning and systems biology, and previously held positions at the University of Edinburgh (Group Leader and Lecturer) and EPFL (Postdoctoral Researcher). He holds a Ph.D. in Computational Biology from the University of Basel and an M.Sc. in Fundamental Physics from Universidad Complutense de Madrid. Throughout his career, he has been recognized with distinctions such as the SIB Award for Best Young Bioinformatician, the Chancellor's Fellowship at the University of Edinburgh, the USIAS Fellowship from the University of Strasbourg Institute for Advanced Study, and the Theory@EMBL Fellowship. His research has been consistently supported by competitive funding bodies, including a recent grant from the Chan Zuckerberg Initiative to explore interpretable deep learning approaches to cancer cell cycle regulation.

Office Hours

Office hours will be on request. Please contact at:

Office hours will be on request. Please contact at: nmolinac@faculty.ie.edu

SUBJECT DESCRIPTION

This course introduces students to the fundamentals of programming using Python. It covers basic programming concepts, control structures, functions, and data structures, providing a foundation for problem-solving and algorithmic thinking. Students will gain hands-on experience by implementing real-world applications using Micro:Bits and exploring robotics and artificial intelligence (AI) concepts with Python libraries. The course emphasizes practical learning through structured lab sessions and a final project.

Computer programming is the art and science of designing and writing software that instructs computers to perform tasks. It is a foundational skill in Computer Science and Artificial Intelligence, underpinning everything from building reliable software systems to developing data-driven and intelligent applications.

This course introduces programming to students with little or no experience. It starts with the basics of how computers run programs (CPU and memory), how code is executed (compiled vs. interpreted languages), and how to plan solutions using pseudocode. Using Python, students then learn how to break problems into steps, design simple algorithms, and write code that is correct, clear, and easy to maintain. Along the way, they practice good habits such as organizing code into reusable pieces and debugging errors systematically.

Learning is hands-on and project-oriented. Through frequent programming exercises, students will apply concepts to problems drawn from computer science and AI contexts, such as data processing, simple simulations, and foundational numerical or vector-based computation. Students will practice translating problem statements into working programs and will gain confidence in iterating on solutions through testing and refinement.

The course provides a comprehensive introduction to fundamental programming constructs, including variables, data types, expressions, and control structures (conditionals and loops). Students will work with core data structures (arrays/NumPy arrays, lists, tuples, sets, and dictionaries), learn modular programming with functions and parameter passing, and develop practical debugging skills to identify and fix errors. The course also introduces the basic Python ecosystem needed for real development, including working with scripts, modules, and files.

By the end of the course, students will have a solid foundation in programming and software development to support subsequent coursework in algorithms, data structures, machine learning, and systems. They will be able to build small-to-medium Python programs that automate tasks, process and analyze data, and implement basic algorithmic solutions—skills that are essential for further study and careers in Computer Science, AI, and related fields.

LEARNING OBJECTIVES

The main objective of this course is to provide students with a strong foundation in programming and computational thinking, preparing them for subsequent coursework in Computer Science and Artificial Intelligence, including algorithms and data structures, data-centric programming, and introductory machine learning.

By the end of the course, students should be able to:

- Write correct, readable Python programs to solve structured and open-ended problems commonly encountered in Computer Science and AI contexts.
- Understand and apply core programming concepts, including variables, data types, expressions, control flow (conditionals and loops), and functions.
- Design simple algorithms and implement solutions using fundamental data structures such as lists, tuples, sets, dictionaries, and basic array-based computation (e.g., NumPy).
- Decompose problems into modular components, define clear interfaces, and reuse code through functions and well-structured modules.
- Apply good programming practices to produce efficient, maintainable, and well-documented code (naming, formatting, basic complexity awareness, and code organization).
- Debug programs systematically by interpreting error messages, using print/debugger workflows, and validating program behavior with simple checks and test cases.
- Work with the Python development ecosystem needed for real projects, including files (text/JSON), libraries, and isolated environments (pip/conda/venv).
- Handle and analyze tabular data with pandas (data cleaning and groupby), visualize results with Matplotlib, and export reproducible outputs (datasets and figures).
- Translate real-world tasks into programs that automate workflows, process data, and support basic computational experiments relevant to AI (e.g., simulation, feature computation, vectorized operations).

In addition, the course will support the acquisition or reinforcement of transversal skills:

- Communicate solutions clearly by summarizing the problem, explaining design decisions, and presenting results in a structured way.
- Develop abstraction skills by identifying patterns, generalizing solutions, and choosing appropriate data representations.
- Collaborate effectively in small teams using basic tooling and practices for shared code (e.g., version control workflows if applicable).
- Build adaptability and independence in learning new tools, libraries, and programming concepts as technology evolves.

TEACHING METHODOLOGY

Each session combines concise theoretical explanations with substantial hands-on programming practice. New concepts are introduced through short lectures and immediately reinforced with guided examples and exercises drawn from Computer Science and AI contexts (e.g., data processing, simulation, algorithmic problem-solving, and vectorized computation). The course emphasizes transfer: students are expected to apply what they learn here to other subjects in the BCSAI curriculum.

Active participation is essential. Students will regularly work on problems during class—individually and in small groups—so they can practice reasoning about code, debugging, and communicating solutions. To maximize practice time, the course follows a flipped-classroom approach when appropriate: students are encouraged to review short readings, videos, or notebook material before class so that in-person time can focus on discussion, live coding, and problem solving.

Learning is supported through three recurring activities:

- **In-class exercises:** frequent short tasks to build fluency and confidence with core programming patterns
- **Problem sets:** deeper, more integrative assignments posted on the campus platform. Students are strongly encouraged to work steadily throughout the term rather than close to exams. Collaboration is encouraged for understanding concepts and discussing approaches, but the intention is that problem-set submissions reflect individual work and provide an honest measure of progress. Selected problems will be discussed in class, and students may volunteer to present solutions (contributing to participation).
- **Brief quizzes:** short, announced quizzes given throughout the semester covering previously taught material. These quizzes help students consolidate knowledge and help the instructor monitor progress and adjust pacing.

Technical requirements: Students must bring a laptop to every session. A Google account with access to Google Drive is required to run Google Colab notebooks used regularly in class. Students will also install and use VS Code as the primary local development environment (installation and initial setup are covered in the course). For consistency and to facilitate collaboration and troubleshooting, students should configure their development tools in English. Some activities may require installing open-source Python libraries locally in a secure and controlled manner.

Learning Activity	Weighting	Estimated time a student should dedicate to prepare for and participate in
Lectures	20.0 %	30.0 hours
Discussions	10.0 %	15.0 hours
Exercises in class, Asynchronous sessions, Field Work	30.0 %	45.0 hours
Group work	20.0 %	30.0 hours
Individual studying	20.0 %	30.0 hours
TOTAL	100.0 %	150.0 hours

AI POLICY

Generative artificial intelligence (GenAI) tools may be used in this course for some specific tasks/assignments with appropriate acknowledgment. If a student is found to have used AIgenerated content inappropriately, it will be considered academic misconduct, and the student might fail the respective assignment or the course.

If you are in doubt as to whether you are using GenAI tools appropriately in this course, we encourage you to discuss your situation with the professor.

Below, is a suggested format to acknowledge the use of generative AI tools. Please note that acknowledging AI will not impact your grade.

I acknowledge the use of [AI systems link] to [specify how you used generative AI]. The prompts used include [list of prompts]. The output of these prompts was used to [explain how you used the outputs in your work]

If AI was permitted to be used in your assignment, but you have chosen not to include any AIgenerated content, the following disclosure is recommended:

No content generated by AI technologies has been used in this assignment.

PROGRAM

MODULE 1/4: FUNDAMENTALS

The course starts with computing basics, the difference between compiled and interpreted languages, and pseudocode to plan solutions. Students then learn core Python syntax and logic: variables and types, if/else and boolean logic, loops, and basic debugging, before moving into functions, scope, and an introduction to generators plus functional tools.

SESSION 1 (LIVE IN-PERSON)

Computing Basics

- The hardware/software interface: CPU, memory, and programs.
- How programs run: compiled (e.g., C) vs. interpreted (e.g., Python) languages.
- Pseudocode: expressing logic before writing code.

SESSION 2 (LIVE IN-PERSON)

Introduction to Python

- Introduction to Python and Google Collab notebooks.
- Variables in Python: numeric (integers and floats), booleans, and string variables.
- Operations with numeric variables (arithmetic operators).
- Operations and methods with string variables (basic text processing).

SESSION 3 (LIVE IN-PERSON)

Flow control

- The if/else statement.
- Logical operators: and, or, not.

SESSION 4 (LIVE IN-PERSON)

Loops and basic debugging

- Looping in Python: for and while loops. Existing a loop using break and continue.
- Combining loops and flow control in Python.
- Introduction to basic debugging using print() and reading error messages (tracebacks).

SESSION 5 (LIVE IN-PERSON)

Functions I

- What is a function? Why modularity matters in software.
- Creating user-defined functions with def.

- Positional arguments, return values.
- Keyword arguments and default values.

SESSION 6 (LIVE IN-PERSON)

Functions II

- What is a docstring? What are type hints? Why they matter for maintainability.
- Function recursion: how a function can call itself.
- Debugging functions: tracing inputs/outputs, minimal tests.

SESSION 7 (LIVE IN-PERSON)

Functions III

- Scope and program reasoning: local/global, lifetime of variables, side effects.
- Iterators & generators: yield, lazy evaluation (streaming/large inputs).
- Functional Programming Basics: lambda, map, filter.

MODULE 2/4: DATA STRUCTURES

In this module, students work with Python's core data structures—lists/tuples, sets, and dictionaries—and practice iterating, indexing, and building data structures efficiently. This module also covers mutability and side effects in functions, and it concludes with an exercise-based review and a midterm exam covering Modules 1–2.

SESSION 8 (LIVE IN-PERSON)

Core data structures and iterations

- Introduction to core data structures and their properties: lists, tuples, sets and dictionaries.
- Iterables in Python: lists and tuples.
- Using indexes to retrieve values from inside an iterable.
- Initializing iterables and changing their contents.
- What is list comprehension?

SESSION 9 (LIVE IN-PERSON)

Sets and membership

- Sets in Python.
- Difference between lists, tuples and sets. How to convert between one another.
- Operating with sets (deduplication, membership testing, filtering).

SESSION 10 (LIVE IN-PERSON)

Dictionaries (hash maps)

- Dictionaries in Python. What are key-value pairs?
- Accessing, adding and removing elements from a dictionary.
- How to create dictionaries from lists, tuples and sets.

SESSION 11 (LIVE IN-PERSON)

Building structures in loops

- How to initialize data structures and update them inside loops.
- Examples and applications with a focus on CS&AI contexts

SESSION 12 (LIVE IN-PERSON)

Mutability and function effects

- How functions may affect data structures: the effects of mutability.Writing safer APIs: “modify in-place” vs “return new object”.

SESSION 13 (LIVE IN-PERSON)

Mid-term review I

- Review of all the contents learnt so far through exercises.
- Examples of Computer Programming applications to CS&AI.
- Group challenge: Be the team to first solve the exercises!

SESSION 14 (LIVE IN-PERSON)

Mid-term review II

- Review for the midterm exam. Covers modules 1 to 2.
- We will do a mock based on exams from the previous years.

SESSION 15 (LIVE IN-PERSON)

MID-TERM EXAM

- Scope: Modules 1 and 2

MODULE 3/4: THE PYTHON ECOSYSTEM

This module focuses on practical development workflow: using the command line and VS Code, working with notebooks in VS Code, and running programs locally. In this module, students learn how to write and execute scripts, organize code into modules and imports, and manage environments and dependencies (conda/pip), with a short, basic introduction to object-oriented programming.

SESSION 16 (LIVE IN-PERSON)

Command Line Interface and Integrated Development Environment

- Introduction to CLI: Terminal fundamentals.
- Introduction to Python IDEs: PyCharm, Spyder, VSCode.
- Downloading and installing VSCode.
- Running code: terminal vs UI; Jupyter notebooks in VSCode.

SESSION 17 (LIVE IN-PERSON)

Scripts, modules and imports

- Writing scripts in Python.
- Running scripts: execution order and the “main” function.
- Code reusability: how to import functions from other scripts.

SESSION 18 (LIVE IN-PERSON)

Environments and dependencies

- Introduction to Python environments.
- Managing environments with Conda.
- How to Install Python libraries with pip and Conda.

SESSION 19 (LIVE IN-PERSON)

Introduction to Object-Oriented Programming

- Programming paradigms: procedural vs. functional vs. object-oriented.
- Why OOP: abstraction, encapsulation, code organization in larger systems.
- Classes and objects: defining a class, creating objects, and using their attributes and methods.
- Codding a mini OOP example

MODULE 4/4: ADVANCED PROGRAMMING

In this module, students cover numerical and data-oriented programming with NumPy. This module also introduces file/data handling, pandas for tabular data, and Matplotlib for visualization and exporting results, leading into the project wrap-up and final review.

SESSION 20 (LIVE IN-PERSON)

NumPy foundations

- Introduction to NumPy.
- Creating and manipulating arrays in NumPy.
- Array indexing and slicing.

SESSION 21 (LIVE IN-PERSON)

Vectorized operations

- Arithmetic operations on NumPy.
- Broadcasting in NumPy.
- NumPy built-in methods and functions.

SESSION 22 (LIVE IN-PERSON)

Shapes and transformations

- Reshaping and resizing arrays.
- Joining and splitting arrays.
- Changing array shapes and dimensions.

SESSION 23 (LIVE IN-PERSON)

Data & file handling for scientific computing

- Paths and folders (relative vs absolute), reading/writing text files.
- CSV + JSON basics (what they are, when to use each).
- Loading/saving data with NumPy.

SESSION 24 (LIVE IN-PERSON)

Basic statistics with NumPy

- Descriptive statistics: mean, median, and standard deviation.
- Correlation and covariance.
- Normalization and scaling as preparation for ML.

SESSION 25 (LIVE IN-PERSON)

Performance and vectorization

- The importance of vectorization: How to write efficient NumPy code.
- Group challenge: Be the team with the fastest code!

SESSION 26 (LIVE IN-PERSON)

Data handling & cleaning with Pandas

- Series vs DataFrame; CSV I/O with pandas..
- Selecting/filtering rows/columns; creating/transforming columns; basic cleaning.
- GroupBy + aggregation.

SESSION 27 (LIVE IN-PERSON)

Data visualization with Matplotlib

- Plotting with Matplotlib.
- Choosing the right plot (scatter/line/hist/bar).
- Save figures + save final datasets.

SESSION 28 (LIVE IN-PERSON)

Project presentations

- Presentations of the group projects (CS/AI-oriented objective).

SESSION 29 (LIVE IN-PERSON)

Course review

- Review for the final exam. Covers modules 1 to 4.
- We will do a mock based on exams from the previous years.

SESSION 30 (LIVE IN-PERSON)

FINAL EXAM

- Scope: All modules 1-4.

EVALUATION CRITERIA

A. CLASS PARTICIPATION

Class participation will be worth 10% of the overall grade. Students are expected to participate actively during lectures with questions and remarks. Class grade will be based on punctuality, participation, and class conduct. There may be a penalty if you create a disruption or talk excessively during class. We will solve exercises from the problem sets during class, and students are encouraged to propose solutions to boost their participation grade.

B. GROUP PROJECT

The group project is worth 15% of the final grade for Computer Programming I. The project is done in collaboration in Linear Algebra, and explores the applications of Computer Programming in that area. Specific details for this project will be provided early in the semester. Students must complete the project and submit it before the final exam. The project includes an in-person presentation, and grading will assess both group collaboration and individual contribution.

C. QUIZZES + MIDTERM EXAM

It is worth 25% of the final grade. We will have several small quizzes scattered throughout the course.

D. FINAL-EXAM

It is worth 50% of the overall grade. You need to score at least 3.5 on the final exam to pass the overall course, even if you have already passed the course through the other course assessments. Information about the detailed characteristics of the final-exam will be given at the beginning of the semester.

criteria	**percentage**	**Learning Objectives**	**Comments**
Final Exam	50 %		You need to score at least 3.5 on the final exam to pass the overall course, even if you have already passed the course through the other course assessments.
Group Work	15 %		The group project will assess students' ability to design and implement a small Python program that solves a concrete problem relevant to CS/AI or data analysis. Grading will consider correctness, code quality, appropriate use of course tools when relevant, reproducibility, and the clarity of the final presentation.

Class Participation	10 %		Each module has its own problem set. The student is expected to work on them and to turn in selected exercises that will be announced throughout the course. Active participation during lectures is expected, with students encouraged to ask questions and make remarks. The class grade will be based on punctuality, participation, and class conduct. Please note that there may be a penalty for disruptions or excessive talking during class.
Quizzes+Midterm	25 %		he midterm exam will cover Modules 1–2. It provides a checkpoint on foundational understanding and problem-solving skills.

** RE-SIT / RE-TAKE POLICY **

Each student has four chances to pass any given course distributed over two consecutive academic years: ordinary call exams and extraordinary call exams (re-sits) in June/July. Students who do not comply with the 80% attendance rule during the semester will fail both calls for this Academic Year (ordinary and extraordinary) and have to re-take the course (i.e., re-enroll) in the next Academic Year.

Evaluation criteria:

- Students failing the course in the ordinary call (during the semester) will have to re-sit the exam in June / July (except those not complying with the attendance rule, who will not have that opportunity and must directly re-enroll in the course on the next Academic Year).
- The extraordinary call exams in June / July (re-sits) require your physical presence at the campus you are enrolled in (Segovia or Madrid). There is no possibility to change the date, location or format of any exam, under any circumstances. Dates and location of the June / July re-sit exams will be posted in advance. Please consider this when planning your summer.
- The June / July re-sit exam will consist of a comprehensive exam. Your final grade for the course will depend on your performance in this exam only; continuous evaluation over the semester will not be taken into consideration. Students will have to achieve the minimum passing grade of 5 and can obtain a maximum grade of 8.0 (out of 10.0) – i.e., "notable" in the re-sit exam.
- Retakers: Students who failed the subject on a previous Academic Year and are now reenrolled

as re-takers in a course will be needed to check the syllabus of the assigned professor, as well as contact the professor individually, regarding the specific evaluation criteria for them as retakers in the course during that semester (ordinary call of that Academic Year). The maximum grade that may be obtained in the retake exam (3rd call) is 10.0.

- After the professor grades ordinary and extraordinary call exams, you will have the possibility to attend a review session for that exam and course grade. Please be available to attend the session in order to clarify any concerns you might have regarding your exam. Your professor will inform you about the time and place of the review session. Any grade appeals require that the student attended the review session prior to appealing.
- Students failing more than 18 ECTS credits (BAM) or 21 ECTS credits (BIEBAM) after the June / July re-sits will be asked to leave the Program. Please, make sure to prepare yourself well for the exams in order to pass your failed subjects.
- In case you decide to skip the opportunity to re-sit for an exam during the June/July extraordinary call, you will need to enroll in that course again for the next Academic Year as a re-taker and pay the corresponding extra cost. As you know, students have a total of four allowed calls to pass a given subject or course, in order to remain in the program.

BEHAVIOR RULES

Please, check the University's Code of Conduct here. The Program Director may provide further indications.

ATTENDANCE POLICY

Please, check the University's Attendance Policy here. The Program Director may provide further indications.

ETHICAL POLICY

Please, check the University's Ethics Code here. The Program Director may provide further indications.