IB 111: Introductory Programming for Biologists
Required Course Information

Course number: IB 111

Course title: Introductory Programming for Biologists

Instructor: John Huelsenbeck

Effective Start Term: Spring 2025

Semester offered, e.g., fall, spring or summer: Fall or Spring

Course description: Unlock the power of programming for biological exploration! This beginner-friendly course introduces biologists to essential programming concepts using C and Python. Dive into hands-on projects tailored for biological applications, equipping you with the skills to analyze data, visualize results, and automate tasks. No prior coding experience required. Embrace the world of programming to elevate your research capabilities in the life sciences!

Grading: Letter or P/NP

Number of units: 3

Final Exam: Written final exam conducted during the scheduled final exam period.

Course format: Three hours of lecture per week.

Prerequisites: None.

Repeatable for credit?: No

Enrollment restrictions: None.

Meeting pattern: Tuesdays and Thursdays for 1.5 hours

Expected enrollment: 100

Justification: Computer programs are used to make sense of biological data, including single cell, sequence, and expression data. In fact, much research in biology relies on the ability of the scientist to stitch together programs in a 'pipeline' of analyses, with the output of one program forming the input for the next. Python is a popular program for constructing such pipelines today, but 20 years ago the scripting language Perl was commonly used to accomplish the same task. In this course, I will teach the students the basics of computer programming in a language agnostic way. The goal is for the student to come away with the skills to understand how to write simple computer programs.

I believe that this course could form an important component of our curriculum and potentially attract a large number of students. It certainly will fill a gap in the data sciences initiative on campus which stresses data analysis but does not seem to stress the more practical aspects of computer programming.

University of California, Berkeley
IB111: Introductory Programming for Biologists
CCN–0000001 — Spring 2025

Instructor Information

Name: John Huelsenbeck
Office: 4153C VLSB
Email: johnh@berkeley.edu
Office Hours: HOURS

Class Information

Dates: DATE START – DATE END
Time: TIME START – TIME END
Classroom: VLSB

Course Description

Imagine the excitement of early biologists when they first peered into microscopes to see the details of cells almost 400 years ago. A comparable excitement surrounds the use of computers in biology which, like the microscope, have provided insights into biological processes. Today, in fact computers and computational methods are a common place in biology, playing a critical role in genome sequencing, functional prediction, similarity matching, and comparative analysis of gene sequences. A savvy biologist understands the computational methods used so the methods can be best applied to his or her data. This course delves deeply into the methods often used, with the main objective of teaching basic programming skills. The student will learn about data types, limits of precision in computer memory, Monte Carlo methods and pseudo random number generation, and some basic data structures such as trees. The course is language agnostic, with the main goal to teach basic principles and introduce features in common to all languages, such as the program control flow statements.

Prerequisites

There are no prerequisites for this course, except for a positive attitude!

Course Learning Objectives

After this course, you should be able to. . .

- To develop a matured perspective on how to turn an algorithm into computer code.
- To improve your understanding of several numerical methods commonly used in computational biology.
- To learn how to describe the efficiency of computer algorithms.

- To learn the basics of computer programming, including data types, functions, control statements, and structures/classes.
- To understand several of the concepts basic to evolutionary biology and population genetics.
- To understand that programming is not just a tool for solving problems but an art and science in and of itself.
- To learn to love computational biology and programming!

Textbook

There are many programming textbooks out there. However, I find myself turning to the internet more often than not to figure out (or remind myself) of how to do many tasks while programming. In this spirit, there is no physical textbook. Rather, the official textbooks for this course are free and available online:

Programming in C by By Bharat Kinariwala and Tep Dobry:
http://ee.hawaii.edu/ tep/EE160/Book/PDF/Book.html

and

Python for Data Analysis, 3E by Wes McKinney: https://wesmckinney.com/book/

Course Help

You're not alone in this course; the instructor is here to support you as you learn the material. It's expected that some aspects of this course will take time to grasp, and the best way to grasp challenging material is to ask questions. You can also contact your instructor using bCourses to set up an appointment outside of office hours, if necessary.

Students with Disabilities

If you require course accommodations due to a physical, emotional, or learning disability, contact UC Berkeley's Disabled Students' Program (DSP). Notify the instructor through course email of the accommodations you would like to use. You must have a Letter of Accommodation on file with UC Berkeley to have accommodations made in the course. UC Berkeley is committed to providing robust educational experiences for all learners. With this goal in mind, we have enabled the ALLY tool for this course. You can now automatically generate "Alternative Formats" for course files and bCourses Pages. Depending on the context, these formats can include Tagged PDF, HTML, BeeLine Reader, Electronic Braille, ePub, Immersive Reader, MP3, and translated versions. For more information, watch the video entitled, "Ally Tutorial for Students at UC Berkeley."

Homework & Labs

There will be a weekly programming assignment that will be assigned on Tuesday and due on the following Saturday (at midnight).

Quizzes

There will be a weekly quiz. The quizzes will be given on Thursdays. The quizzes will cover the material covered in the lectures since the last quiz. This a chance for you to see if you are mastering

the course material. Start preparing for quizzes before they arrive! Quizzes will be closed book and closed notes.

Exams

There is a midterm and a final exam. You are expected to be present, seated, and ready to take the exam before the exam begins. You are not permitted to use any outside materials, resources, or electronic devices (including but not limited to mobile phones, smartwatches, etc., but not including a calculator) on the exams. Any violation of this policy is a violation of the university's Academic Integrity Policy.

The final exam is comprehensive and will be given during the final exam period on DAY, YEAR between [TIME] – [TIME]. The location will be announced later in the semester.

Late Quizzes, Assignments, and Exams

The lowest Thursday quiz score will be dropped, so you can miss one quiz without much of an impact on your final grade. Accommodations, of course, will be made for religious holidays or medical reasons.

Grading

The course grade is determined by the following components:

Midterm	25%
Homework	25%
Quizzes	25%
Final Exam	25%

Grade Scale

Final grades will be assigned according to the following scale:

A	93 – 100	C+	77 – 79
A–	90 – 92	C	73 – 76
B+	87 – 89	C–	70 – 72
B	83 – 86	D	60 – 69
B–	80 – 82	F	0 – 59

Academic Integrity

You're a member of an academic community at one of the world's leading research universities. Berkeley creates knowledge that has a lasting impact in the world of ideas and on the lives of others; such knowledge can come from an undergraduate paper as well as the lab of an internationally known professor. One of the most important values of an academic community is the balance between the free flow of ideas and the respect for the intellectual property of others. Scholars and students always use proper citations in papers; professors may not circulate or publish student papers without the writer's permission; and students may not circulate or post materials (handouts, exams, syllabi—any class materials) from their classes without the written permission of the instructor.

Any test, paper or report submitted by you and that bears your name is presumed to be your own original work that has not previously been submitted for credit in another course unless you obtain prior written approval to do so from your instructor. In all of your assignments, including your homework or drafts of papers, you may use words or ideas written by other individuals in publications, websites, or other sources, but only with proper attribution. If you're unclear about the expectations for completing an assignment or taking a test or examination, be sure to seek clarification from your instructor beforehand. For additional information, read this UC Berkeley Library guide on How to Avoid Plagiarism. As a member of the campus community, you're expected to demonstrate integrity in all of your academic endeavors and will be evaluated on your own merits. The consequences of cheating and academic dishonesty—including a formal discipline file, possible loss of future internship, scholarship, or employment opportunities, and denial of admission to graduate school—are simply not worth it. Read more about Berkeley's Honor Code.

The elephant in the room involves generative AI, such as ChatGPT. Such technology can complete many of your assignments much more quickly than you ever could. And, they would typically do a good job. That said, many programming tasks are beyond the ability of generative AI. Believe me, I've experimented with ChatGPT for many of the programs that I write as part of my research and found the results disappointing. Perhaps more importantly, you will not gain any practice or confidence by writing the code yourself. You will certainly not be in a position to do well on the midterm and final exams which are closed book and closed internet.

Incomplete Course Grade

Students who have substantially completed the course but for serious extenuating circumstances, are unable to complete the remaining course activities, may request an Incomplete grade. This request must be submitted in writing to the instructor. You must provide verifiable documentation for the seriousness of the extenuating circumstances. Refer to the Office of the Registrar's website for more information on the university's policy on Incomplete Grades.

Important Dates

- Financial Aid/Academic Drop Deadline: DATE
- Midterm: DATE
- Academic Withdrawal Deadline: DATE
- Final Exam: DATEs

Respect Policy

I respect your time:

- I will come prepared to help you understand the course material and prepare you for quizzes/exams.
- Communication is key: I cannot help you if I do not know what is going on.
- I am here to help you, this is your time, so let me know what I can do to help you succeed.
- If there is something that you would like me to do differently, please, let me know. I am happy to work with you to make class the best it can be.

Respect my time:

- Be on time to class.
- Pay attention when I am talking to you.
- Come to class prepared by doing the work and going to office hours when you need help.

Respect each other:

- Do not be disruptive. If you need to take a call or text someone, take it outside.
- Work with each other to find solutions. You learn more by helping each other.
- Allow one another to make mistakes. This is an important part of the learning process.
- Use respectful language when talking with one another.

Tips for Success

- Be proactive about your success in the course.
- Do not procrastinate! Begin your assignments and studying early!
- Attend every class.
- Ask questions during lecture.
- Form a study group! Working together will help you and others better understand the course material as you can work through different difficulties and offer each other clarifications on concepts.
- Always check to be sure that you understand a concept.
- Every time you approach a new concept, carefully think how it could be applied.

Syllabus[1]

Weeks 1–4: Introduction to Programming with C

Lecture 1: Course Overview and Introduction to C

- Overview of the course structure.
- Introduction to the C programming language.
- Setting up the C development environment.
- **Reading:** K&D 1.1
- **Assignment 1:** Write a simple C program to print "Hello, World!" to the console.

Lecture 2: Basic C Syntax and Data Types

- Variables, data types, and constants in C.
- Basic input and output operations.
- **Reading:** K&D 1.2
- **Assignment 2:** Create a C program that takes user input for two numbers, performs basic arithmetic operations, and displays the result.

Lecture 3: Control Flow in C

- Conditional statements (if, else, switch).
- Looping structures (for, while, do-while).
- **Reading:** K&D 2
- **Assignment 3:** Implement a C program that checks if a given number is prime or not.

Lecture 4: Functions in C

- Function declaration and definition.
- Parameters and return values.
- Function prototypes.
- **Reading:** K&D 3
- **Assignment 4:** Write a C program to calculate the factorial of a given number using a function.

Lecture 5: Arrays and Pointers in C

- Working with arrays and pointers in C.
- Understanding memory addresses.
- **Reading:** K&D 6 & 7
- **Assignment 5:** Create a C program to find the sum and average of elements in an array.

[1] Syllabus is modeled on a Tuesday/Thursday lecture schedule. Note that the reading assigned to each lecture will be fine-tuned once the lectures are more fully developed. I anticipate assinging additional readings for some topics.

Lecture 6: Structs and Memory Allocation in C

- Defining and using structs.
- Dynamic memory allocation and deallocation.
- **Reading:** K&D 12
- **Assignment 6:** Develop a C program that manages a student database using structures and dynamic memory allocation.

Lecture 7: File Handling in C

- Reading and writing files in C.
- File input/output operations.
- **Reading:** K&D 13
- **Assignment 7:** Implement a C program that reads data from a file, performs some computations, and writes the results to another file.

Lecture 8: Introduction to Algorithms and Sorting in C

- Basic algorithms and their analysis.
- Sorting algorithms in C.
- **Reading:** K&D 9
- **Assignment 8:** Implement a sorting algorithm in C (e.g., bubble sort, insertion sort) and apply it to a set of data.

Weeks 5–8: Advanced C Programming for Scientific Computing

Lecture 9: Advanced Pointers and Dynamic Memory Allocation in C

- Pointers to functions.
- Advanced memory management techniques.
- **Reading:** K&D 8
- **Assignment 9:** Develop a C program that simulates a dynamic data structure (e.g., linked list, tree) using pointers.

Lecture 10: Recursion in C

- Understanding recursive functions.
- Solving problems using recursion.
- **Reading:** No reading
- **Assignment 10:** Write a C program to calculate the Fibonacci sequence using recursion.

Lecture 11: Data Structures in C for Scientific Computing

- Linked lists, stacks, and queues.
- Implementing basic data structures for scientific applications.
- **Reading:** TBD

- **Assignment 11:** Implement a data structure in C that is suitable for storing and processing scientific data.

Lecture 12: Multidimensional Arrays in C

- Working with matrices and arrays in a scientific context.
- Operations on multidimensional arrays.
- **Reading:** K&D 11
- **Assignment 12:** Develop a C program that performs matrix multiplication and demonstrates its use in scientific calculations.

Lecture 13: File Handling for Scientific Data in C

- Reading and writing data files.
- Handling different file formats.
- **Reading:** K&D 13
- **Assignment 13:** Create a C program that reads data from a file, processes it to extract relevant scientific information, and outputs the results.

Lecture 14: Random number generation

- Basics of pseudo-random number generation.
- Transforming uniform(0,1) random numbers.
- **Reading:** TBD
- **Assignment 14:** Write a program that generates exponential random numbers.

Lecture 15: Trees

- Understanding trees
- Pre- and postorder traversal of trees
- **Reading:** TBD
- **Assignment 15:** Write a program that generates a tree under the birth-death process of cladogenesis.

Weeks 9–12: Introduction to Scientific Computing with Python

Lecture 16: Introduction to Python Programming for Scientific Computing

- Overview of the Python programming language.
- Setting up the Python development environment.
- **Reading:** M 1
- **Assignment 16:** Write a simple Python script to calculate and display the factorial of a given number.

Lecture 17: Basic Python Syntax and Data Types

- Variables, data types, and basic operations in Python.

- **Reading:** M 2
- **Assignment 17:** Create a Python program to read user input, perform string manipulations, and display the results.

Lecture 18: Control Flow in Python

- Conditional statements (if, elif, else).
- Looping structures (for, while).
- **Reading:** M 2
- **Assignment 18:** Develop a Python script that checks if a given year is a leap year or not.

Lecture 19: Functions and Modular Programming in Python

- Defining and calling functions in Python.
- Scope and lifetime of variables.
- **Reading:** M 2
- **Assignment 19:** Write a Python program that uses functions to perform a series of mathematical operations.

Lecture 20: NumPy for Scientific Computing in Python

- Introduction to NumPy for numerical operations.
- Working with arrays and matrices.
- **Reading:** M 4
- **Assignment 20:** Use NumPy to implement a Python program that performs basic linear algebra operations on matrices.

Lecture 21: Scientific Data Visualization with Matplotlib

- Basic plotting and visualization techniques.
- Creating plots for scientific data.
- **Reading:** M 9
- **Assignment 21:** Create a Python script that visualizes scientific data using Matplotlib.

Lecture 22: Introduction to Pandas for Data Analysis in Python

- Working with data frames in Python.
- Data manipulation and analysis.
- **Reading:** M 5
- **Assignment 22:** Perform basic data analysis on a provided dataset using Pandas.

Weeks 13–15: Advanced Scientific Computing with Python

Lecture 23: Introduction to Scientific Libraries in Python

- Overview of libraries like SciPy for scientific computing.
- Using pre-built functions for common scientific tasks.
- **Reading:** M 1
- **Assignment 23:** Apply SciPy functions to solve a scientific problem, such as numerical integration or solving differential equations.

Lecture 24: Symbolic Computing with SymPy

- Introduction to symbolic mathematics in Python.
- Solving equations symbolically.
- **Reading:** M 1
- **Assignment 24:** Use SymPy to solve and manipulate mathematical expressions symbolically.

Lecture 25: Scientific Machine Learning with scikit-learn

- Basics of machine learning for scientific applications.
- Using scikit-learn for predictive modeling.
- **Reading:** TBD
- **Assignment 25:** Apply scikit-learn to a provided dataset for a basic machine learning task, such as classification or regression.

Lecture 26: Integration of Python with External Libraries

- Using external libraries and APIs for scientific applications.
- Integration with Fortran and C libraries.
- **Reading:** M 12
- **Assignment 26:** Integrate a Python script with an external scientific library (e.g., a Fortran or C library) to extend functionality.

Lecture 27: Advanced Data Visualization with Plotly and Seaborn

- Creating interactive plots and advanced visualizations.
- Presenting scientific results.
- **Reading:** M 9
- **Assignment 27:** Develop a Python script that uses Plotly or Seaborn to create advanced visualizations for scientific data.

Lecture 28: Introduction to Parallel and Distributed Computing in Python

- Basics of parallel and distributed programming in Python.
- Using parallelism for scientific computations.
- **Reading:** TBD

- **Assignment 28:** Parallelize a Python script for a scientific computation using parallel processing or distributed computing.

Lecture 29: Final Python Project for Scientific Computing

- Students work on a small Python programming project related to scientific computing.
- **Reading:** No reading
- **Assignment 29:** Develop a Python program that addresses a real-world scientific problem, applying concepts learned throughout the course.

Lecture 30: Course Review and Future Directions in Scientific Computing

- Recap of key concepts.
- Next steps and resources for further learning in scientific computing.