Bi 410/510: Introduction to Programming for Biologists

Spring 2019

Biologists who collect and analyze data often develop a complicated "workflow" that involves several different steps. In order to get data from one program to another, scientists find themselves reading through large text files, copying and pasting between documents, and often reformatting the data for the next application. That's not only tedious, it's error-prone. These operations should be automated as much as possible.

This class is a hands-on introduction to the practical skills required to work with large data sets. We'll start by learning to use a **command line interface**, where users type out the instructions that tell the computer which program to run and where to save the output files.

The course continues with basic concepts in **Python programming**: breaking complex tasks into manageable pieces, reading and writing data files, and using iteration ("loops") and other control structures. We will also see how to find code libraries and use software written by others, in particular modules written specifically for biology applications.

Most of the projects will use **Jupyter Notebooks** to create "living documents." A notebook can contain Python code and documentation that explains what the code does and how it works. When a person using the notebook runs the code in the notebook the results are inserted automatically into the notebook. We'll learn how to use Python libraries like `pandas` (for statistical analysis) and `matplotlib` (for data visualization) and see how the tables and charts produced by these libraries are displayed directly in the notebook.

We'll finish the term by learning how to save shell commands in a simple **script** so the operations can be performed automatically. Python is also a great scripting language, and we'll see how to write Python scripts so we can develop fully automated computational workflows.

Course Information:

Instructor:	John Conery `conery@uoregon.edu`
GE:	Murillo Fernando Rodrigues `murillor@uoregon.edu`
Office Hours:	John: Mon 1–2, Wed 10–11 (281 Onyx) Murillo: Tue 11:30–12:30, Thu 9–10 (276 Onyx)
Lectures:	MW 4:00 – 5:20, 202 Cascade
Lab:	F 2:00-3:50, 130 Huestis
Textbook:	*Practical Computing for Biologists*, by Steven Haddock and Casey Dunn, `http://practicalcomputing.org`.
Prerequisites:	None (no prior computer programming experience is necessary)

Grading

Grades will be based on the total number of points earned during the term. There are three ways to earn points:

- short **programming projects** that introduce computing skills and give students a chance to practice using those skills
- a series of self-paced **milestone exams** that can be taken in lab sessions
- short **in-class exercises**, some of which may involve pre-class reading assignments

Programming Projects (80 Points)

There will be eight short projects, assigned (approximately) one per week, starting with the first week.

For these projects students are encouraged to work in small groups of two or three people. We will check each project when it is submitted. If the work is satisfactory, each person in the group will earn 10 points. If the work is not satisfactory the group can revise and resubmit a new version, correcting any issues identified by the graders.

There is no due date for projects, but projects are prerequisites for milestone exams and must be completed before taking the corresponding exam.

Milestone Exams (150 Points)

Milestone exams are designed to test how well a student knows the main concepts:

- shell commands (Unix command line interface)
- basic Python expressions and commands
- working with lists in Python
- reading and writing files with Python
- program design, defining functions
- data analysis with `pandas`
- data visualization with `matplotlib`
- using Python as a scripting language

Exams will be administered during lab. The first part of the weekly lab period (Fridays from 9:00 to 11:00 or 11:00 to 1:00) will be a general discussion and work session. After that students can take a milestone exam if they want, or continue working on their own or with group members on other projects.

The exams are self-paced – a student can take an exam at any point during the term, once they have completed the corresponding programming project. A student can repeat an exam, up to a maximum of three attempts, and we will use the highest score.

Note that some exams might have "due dates," meaning they must be completed by that date (so we can discuss the exam and the results during lectures after everyone has had a chance to take it).

For more information about these exams see the FAQ section on the Canvas web page for this course.

In-Class Exercises (20 points)

Throughout the term there will be other opportunities to earn points by participating in group projects. Some exercises will be based on reading assignments, others will be in-class group programming exercises.

Exercises will be announced ahead of time during class and posted on Canvas.

Final Exam Period

There will be no midterm exam or final exam. The final exam period for this term (2:45 to 4:45 on Thursday June 13) will be an open period where students can take any milestone exams they have not yet completed.

Final Grades

This table shows the number of points that can be earned for each project and milestone exam:

Topic	Unit Name	Project Points	Milestone Exam Points
Shell Commands	shell	10	10
Intro to Python	python	10	20
Data Structures	lists	10	20
Reading and Writing Files	io	10	20
Functions	func	10	20
Data Analysis	pandas	10	20
Visualization	vis	10	20
Python Scripts	scripts	10	20

In-class exercises will be worth a total of 20 points, so the total number of points possible is 250: 80 from group projects, 150 from milestone exams, and 20 from class exercises.

Letter grades will be assigned based on the number of points a student has earned throughout the term. The "comment" column in this table gives an example of how a student could earn a particular total.

Total	Grade	Comment
below 115	D	completed all group projects but did not pass any milestone exams
115–129	C−	
130–144	C	completed all projects, minimum score on all milestones, some participation
145–164	C+	
165–179	B−	
180–194	B	all projects, working programs on all milestones, full participation
195–214	B+	
215–224	A−	
225+	A	all projects, good style and documentation on milestones, full participation

Learning Outcomes

I = introduce
D = develop
M = master

(1) Know how to use a terminal emulator and command line interface [D/M]

(2) Know how to implement simple functions in Python [D]

(3) Understand basic data structures and control flow in Python, how to apply them [D]

(4) Learn general techniques for developing, testing, and debugging software [I/D]

(5) Be able to find, install, and use Python modules [D]

(6) Basic knowledge of Pandas (Python's data analysis library) and MatplotLib (Python's data visualization library) [I]

(7) Be able to write simple shell scripts and use Python to automate workflows [I]

(8) Know how to use an SQLite database to store project data [I]

Students should know about the command line interface for several reasons: it provides useful background for writing Python programs that read and write files, it's required for writing scripts (programs that control other applications, *e.g.* as part of an analysis pipeline), and it's necessary for running jobs via queuing systems on high performance computers.

The "Python Literacy" goal for this course is learning about the fundamental data structures – strings, lists, and associative arrays (called "dictionaries" in Python) – and how to use them. Students will write a few simple programs that use these objects.