# Bi 410/510: Introduction to Programming for Biologists

**Winter 2018**

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 automate data analysis workflows. 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. Next we'll see how to save commands in a simple **script**, so the same operations can be performed automatically.

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. Python is also a great scripting language, and we'll see how to write Python scripts so we can develop fully automated computational workflows.

We will also see how to find code libraries and use software written by others, in particular modules written specifically for biology applications. If time permits, there will be projects using **databases** and Python libraries for **statistical analysis** and **data visualization**.

**Course Information:**

| | |
|---|---|
| *Instructor:* | John Conery<br>`conery@uoregon.edu` |
| *GTF:* | Alex Gutierrez<br>`agutier4@uoregon.edu` |
| *Office Hours:* | times and locations will be posted on Canvas |
| *Lectures:* | MW 4:00 – 5:20, 252 Straub |
| *Lab:* | F 2:00-3:50, 129 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 (16 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 2 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 some projects are prerequisites for a milestone exam and must be completed before a student takes the exam (see the table below).

### Milestone Exams (30 Points)

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

- text files and data formats
- shell commands (Unix command line interface)
- basic Python expressions and commands
- working with lists in Python
- reading and writing files with Python
- writing scripts with Python
- an advanced topic TBD (*e.g.* data analysis or data visualization)

Exams will be administered during lab. The first hour of the weekly lab period (Fridays from 2:00 to 4:00) will be a general discussion and work session. In the second hour students can take a milestone exam if they want, or continue working on their own or with group members on other projects. Exams must be completed on one of the department laptops in the Huestis lab (room 129).

The exams are entirely self-paced – a student can take any 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.

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

**In-Class Exercises (4 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 Tuesday March 20) 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 avaialable for each project and milestone exam:

| Topic | Unit Name | Project Points | Milestone Exam Points |
|---|---|---|---|
| Text Files | text | 2 | 1 |
| Shell Commands | shell | 2 | 4 |
| Intro to Python | python | 2 | 5 |
| Data Structures | lists | 2 | 5 |
| Reading and Writing Files | io | 2 | 5 |
| Python Scripts | scripts | 2 | 5 |
| Compute Severs | remote | 2 | – |
| TBD | advanced | 2 | 5 |
| Total | | 16 | 30 |

In-class exercises will be worth a total of 4 points, so the total number of points possible is 50: 16 from group projects, 30 from milestone exams, and 4 from class exercises.

Letter grades will be assigned based on the number of points a student has earned throughout the term.

| Score | Grade | Comment |
|---|---|---|
| 15 | D | completed all group projects but did not pass any milestone exams, some participation in class |
| 21 | C− | all projects, earned mininum score on all milestones, some participation |
| 23 | C | |
| 26 | C+ | all projects, working programs on half of the milestones, full participation |
| 32 | B− | |
| 38 | B | average 3 points on exams or maximum score on half of them |
| 40 | B+ | |
| 42 | A− | |
| 44 | A | average 4 points on exams |
| 50 | A+ | |

## 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 in Python, how they are used [D]

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

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

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

(7) Know how to use R or Pandas (a Python module) for analyzing and plotting 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.

SQL and R are both valuable for biology projects. Time permitting, these languages will be introduced in the middle of the term and used along with Python for projects in the second half.