Spring 2020

# Bi 410/510

## Introduction to Programming for Biologists

Biologists who collect and analyze data often develop a complicated "workflow" that involves several different steps. Researchers find themselves reading through large text files, copying and pasting between documents, and reformatting data to be used in different applications. These steps are not only tedious, they can be error-prone. Scientific workflows should be automated as much as possible.

This class is a hands-on introduction to the practical skills required to develop workflows for 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 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.

## Course Information

All lectures, lab sessions, and office hours will be online meetings managed with Zoom.

| | |
|---|---|
| *Instructor:* | John Conery (`conery@uoregon.edu`) |
| *GE:* | Victoria Caudill (`vcaudill@uoregon.edu`) |
| *Office Hours:* | John: Mon 1–2, Wed 10–11, or by appointment |
| | Victoria: TBD |
| *Lectures:* | MW 4:00 – 5:20 |
| *Labs:* | F 9:00 – 10:50 |
| *Textbook:* | *Practical Computing for Biologists*, by Steven Haddock and Casey Dunn `http://practicalcomputing.org`. |
| *Prerequisites:* | None (no prior computer programming experience is necessary) |

## Canvas

We will use Canvas to distribute projects and milestone exams (described below).

Students will upload completed projects and exams through Canvas, and we will post scores and feedback on each submitted item.

Other things posted on Canvas include:

- instructions for installing software used in the course
- links to on-line references for Python and other software
- an FAQ document with answers to common questions

**Important:** We will also post announcements on Canvas, so make sure you check Canvas regularly (or better yet configure the system so you are automatically notified when there is a new announcement).

## 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** 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.

Projects are prerequisites for milestone exams and must be completed before taking the corresponding exam.

### Milestone Exams (150 Points)

Milestone exams test how well a student has learned the concepts and skills from a programming project. Exams are not group projects; each student must take their own exam.

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.

**Note:** A student can repeat an exam, up to a maximum of three attempts, and we will use the highest score.

Exams will be posted on Canvas as timed quizzes. Students will have 90 minutes to complete the quiz and upload their solution to Canvas.

For more information about milestone exams see the FAQ section on Canvas.

### 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.

### Deadlines

This course is entirely self-paced. There are no due dates during the term. Groups can submit (or re-submit) projects at any time, and students can take exams whenever they are ready.

If a group submits a project by Wednesday at 5:00, we will grade the project and provide feedback so the group members can ask questions during that week's lab session.

The final date for submitting projects is Thursday, June 11, at 4:45 P.M. (the end of the final exam period scheduled for this class).

## Final Grades

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

| Topic | Unit Name | Project | Milestone Exam |
|---|---|---|---|
| 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 |
| **Total** | | 80 | 150 |

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.