CSCI 121: Computer Science Fundamentals I

Spring 2025

Instructor: Jim Fix (jimfix@reed.edu)

Class schedule: This class has two kinds of meetings. They are:

- **Lecture:** Monday 1:10-2:30 in Library 204. This is where we meet to discuss the course material, including our study of Python constructs and sample Python code, and also some CS concepts.
- **Lab: Tuesdays.** This is where we work together in Python and start each weekly homework. For each section, the meeting times are:
    - **Section S01:** Tuesday, 10:30-11:50 in Library 340
    - **Section S02:** Tuesday, 1:40-3:00 in Library 340

Office hours: these are subject to change, but Jim has set aside hours when he can be reached for questions and for help on Mondays and Wednesdays 11:20-12:20 in Library 314. He is also available most Tuesday and Wednesday mornings, most late afternoons Monday, Tuesday, and Wednesday, and other times by appointment.

Tutoring hours: Once the semester is rolling, we will have drop-in tutors available Sunday through Thursday evenings, 7:00-9:00 in Library 340. You can also arrange one-on-one tutoring through the Dojo.

Website: The course uses a web page from Jim's professional pages hosted by GitHub, accessible at jimfix.github.io/csci121. All course materials will be there. Lab homework and projects will be submitted through gradescope.com. We'll also post graded quizzes and exams on Gradescope.

Resources: The primary reference for the course are Jim's slides and other lecture materials. Everything you need to complete assignments should be covered in these. It will also be useful to reference the course notes that Professor Adam Groce has written for past versions of this course. These are available for free on the course website (but we ask that you not distribute them to others). You might also find the textbook *Composing Programs* by John DeNero useful, particularly Chapters 1 and 2. That book is available for free online at composingprograms.com. Another good text is *Think Python!* by Allen Downey, available at greenteapress.com/wp/think-python-3rd-edition. All three of these texts cover the material we cover, in somewhat different order and with some differences in emphasis. They each can serve as a useful supplement to what we cover in lecture. It would be good to make a habit of consulting one/all of them from time to time throughout the semester for their treatment of the course topics.

What we learn in this course

The goal of this course is to introduce you to the principles of computation. That includes teaching you to write a program, as well as how a computer interprets and runs that program. Writing a program well can be a difficult skill to master. You must understand the tools available to you, and understand the principles of good design that allow them to be used in manageable, understandable ways.

Our vehicle for building these skills will be programming in Python, which is a good introductory language for a variety of reasons. First, it uses a reasonably simple syntax, reducing the need to get bogged down in messy details. Second, it is very flexible, allowing us to show you several different ways of writing programs. Finally, it's a widely used language, meaning that knowing it will be potentially quite useful to you after the course.

Given how much time we will spend programming in Python, you could be excused for thinking this was simply a course in how to write Python programs. It is not. Python is a language that shares many constructs with lots of other useful programming languages, so learning its language will make it easy to learn others. The programming tools we use will make your learning of other programming tools easier. There are some areas of programming (like graphical interfaces) that are very useful for practical applications, but which do not require much new conceptual understanding. The course should make it easy for you to learn those things on your own with some quick internet searches.

This course is also meant to be an introduction to computer science as a field. Most areas of computer science use programming as a tool, which is why we begin the study with a programming course. We cover some rich ideas that wholly inform computing as a science when we discuss, for example, recursion, the efficiency of data structures and algorithms, and the schemes a tool like Python uses to reason about, check, and execute your programs' code.

Responsibilities

Lab homework: Lab programming assignments are your weekly work in this class, used to practice the new material you learn. It will be assigned each Tuesday for the lab, and the first several problems assigned each week will be completed in the lab meeting, either as a group or with a partner. The rest must be done outside of class and will be due early the next Tuesday morning.

Projects: There will be four larger programming projects. These involve bigger blocks of code than the homework and give you some experience with programming tasks that are richer, and more involved. They'll often give you a little room to be creative and to decide for yourself exactly how you want your program to behave. These projects can take considerable time, and you will be working on them at the same time you are completing regular homework. It is important that you manage your time and not put them off too long**.**

Exams and Quizzes: Every so often at the beginning of lectures there will be a 10-minute quiz. We need to work hard to switch from quiz to lecture, so you need to make sure you are in class on time. In addition, there will be two in-class exams, 80 minutes in length, that will each cover several weeks of material. Lastly, there will be a comprehensive, final exam scheduled by the Registrar for finals week.

Quizzes and exams will be in written form. You'll write code on paper without access to a computer or to your course notes, and you'll have limited time to do so. This may seem strange to some of you, but we believe your skills in reading, writing, and reasoning about computer code will be strongest when you internalize the programming constructs you are learning, become fast at developing your ideas, and learn to detect problems in your code (and correct them) without the aid of a computer.

Grading: Your grade will be determined by weighting the components listed above, roughly as follows:

- **Lab and homework exercises:** 20%
- **Projects:** 25%
- **Quizzes:** 10%
- **Exams:** 25%
- **Final exam:** 20%

Learning outcomes

The overview above is probably more informative, but every class at Reed also has official learning objectives. Those objectives for this course are the following: Students will be able to

- write moderately complex programs using a high-level programming language. This programming practice includes
    - program decomposition into functions, procedures, methods, and classes
    - console and file input and output
    - conditional execution, iteration, and exception handling
    - object-oriented programming
    - recursive functions
    - higher-order functions
    - basic data structures, including link-based data structures
    - use of existing libraries
- reason about what will happen when code is executed
- write code quickly and easily without the help of computer tools or reference materials
- use programming tools to test, debug and improve larger, more complex pieces of code
- analyze the running time of simple programs
- understand the design and running time of several standard search and sorting algorithms
- organize and document their code
- think creatively and logically about challenging or puzzling problems and apply appropriate strategies to that effort, including persisting through difficulty, working with others, and asking for help when appropriate

Other policies

Attendance: Attendance of lecture and lab is mandatory. We will automatically excuse five absences. Each unexcused missed day will result in a point deduction in your grade. If you need more than five absences to be excused, we will ask for documentation.

Submitting work: Lab homework and projects will be submitted online using Gradescope and some of it will be "autograded" using test procedures we'll configure on Gradescope. This will give you feedback as to whether your code appears to be working. There can sometimes be technical hiccups with the autograder system. If you see any problems with the system, do not hesitate to contact us.

The autograder cannot possibly test your code exhaustively for most problems, and furthermore you should not be solely using the autograder to see if your code works. Test it on your own, trying it out under several scenarios. Thinking about all ways that a program should work, or might not work, is useful in helping you understand its code as you develop it.

Clarity in programming: When you write code, you are writing code that needs to be understood by others (us and the graders). That means writing code in understandable ways and documenting that code well. We will talk in class about how to do this, and your grade will depend on how well it has been done. It is not enough for the program to do the correct thing. In addition to our automated testing, for some homework and especially for the projects, we will give you feedback on the quality of your coding, not just on whether or not it worked and worked well.

Relatedly, it's often possible to write code using Python tricks and features that we do not teach in class. While learning Python beyond what's taught in the course is useful and fun, be careful when you do so. Most of the time our exercises are asking you to solve puzzles using only a few tools, and we might purposely restrict what you can use to solve a problem. This kind of puzzling and problem solving strengthens your skills as a programmer and makes a lot of the programming a fun challenge.

Academic integrity: All individual work you turn in for this class should be yours and yours alone. We take this very seriously and will not hesitate to report violations of this principle. Working on exercises together can be great, and we encourage you to do it. You are allowed to discuss exercises with other students and even help a classmate find a bug in their code. You are not allowed to give code to each other or work on each other's code. You *are* allowed to work together to write code collaboratively, but if so you must list your collaborators in the submitted file. (And you must actively be working together – we expect you to know the difference between collaboration and giving someone an answer.) If you have any question about whether some level of cooperation is acceptable, ask.

For some lab exercises and for the final project, you will be invited to collaborate with another student in this class. In those cases, the same rules above apply to your duo as a single team completing the work.

You are also allowed to look up general background on the internet. This can include looking up rules of Python syntax, for example. You *cannot* use the internet to find code that is intended to solve the problem you are trying to solve, or to copy such code. As a rule of thumb, ask yourself "If this exercise was completely different but still on the same topic, would this resource still be just as useful to me?" If the answer is no, it is not an acceptable resource to use.

Forbidden internet use includes use of artificial intelligence programs like ChatGPT or Github copilot. These tools can be useful as an experienced programmer, but at this stage in your education they would prevent you from learning important skills. You are also not allowed to consult materials from prior offerings of this course.

On quizzes, exams, and the final, of course no communication at all is allowed with the internet or any other person other than the instructor.

Advice

Don't procrastinate! It is very hard to predict how much time a given program will take to write, even a very simple one. Sometimes what seems easy will turn out to be hard, and what seems hard will turn out to be easy. Often a program feels 99% done, with only a couple little things remaining, and that final 1% ends up taking as long as the first 99% took. You might find that you don't understand something you thought you did and need to ask questions in lab or office hours. Leave extra time. Most of the time you won't need it, but sometimes you will.

Prioritize understanding. Your grade in this class is based quite heavily on the exams and quizzes, much more than on your work on the lab exercises, even though those exercises may take up the most significant portion of the time you commit to this course. Nevertheless, your effort on these exercises can significantly impact your grade. It is expected that most students will complete nearly all the lab homework exercises. Completing all these will help you learn the material, and demonstrate your understanding on the exams, quizzes, and projects. You should prioritize completing them in ways that help you gain a good understanding of the material. You can do this in a low-impact way by asking for help from us when you need it, and by discussing approaches to them in lecture and in lab with your instructor and with your classmates. But do not seek help to the extent that you don't force yourself to figure things out.

In this class (and college and life in general) you will benefit greatly from strong *metacognition*, meaning thinking-about-thinking. You should reflect on how you feel about different topics and try to honestly assess your own level of understanding. You should focus your efforts there.

Learn to code without a computer. You should make it a goal to write and check your code without having the computer sitting in front of you. This is a critical skill for communicating your work and ideas to other people. It also gets you into the habit of anticipating what will happen when your code runs, especially any bugs that might be lurking in that code. The quizzes and exams are designed to strengthen

this practice, but you might consider, at times, sketching out your homework and programs in this way—on paper, say, away from the computer—as you go through the course.

Get help. There is a *lot* of help available to you in this class. You can ask for help in lab of course, but also in office hours or evening lab hours. You can request weekly individual tutoring from the DoJo. These are not resources intended only as a last resort for struggling students. They are intended to be a standard part of the course for everyone. You will succeed best if you use them.

Don't get frustrated! Programming and computer science in general require thinking in ways that will be new to many of you. That's part of why this is such a valuable class to take and why it can be so much fun, but it can also make it very difficult, especially at first. Sometimes things will click easily and sometimes they won't. Sometimes you get code mostly working very quickly, and then that last little bit of polishing takes hours and hours. Don't expect everything to work out perfectly the first time. It is entirely normal to run into obstacles along the way.

Be a good community member. Remember that your actions are part of what makes this course a good or bad experience for everyone else. Help each other out. Respect the expectations of academic integrity so that we can continue to organize this course in a way that places a lot of trust in individual students. Remember also that everyone is coming into this course from a different place. Some of you are computer hobbyists who have written some code before. Some of you may have seen some code in another class or working in a science lab. Some of you are completely new to coding and have never worked seriously with computers before. This class is meant for everyone, and we're trying to bring everyone from many different starting points to the same endpoint. You might find some topics very easy and others wildly difficult. Other people might feel the same, but with a different set of challenging topics. We're all in this together.

Have fun! Learning to work with computers can be a fantastic experience. It changes mysterious objects you just take for granted into something you can understand. Whatever you want to do outside computer science, whether it's biology or art or sociology or economics, programming can help make it happen. And most importantly, it's just extremely interesting and intellectually rewarding. Don't get so lost in the details of the work that you don't enjoy the experience.